Securing End-to-End Encrypted Systems

Virginia Claire Blessing

King's College

May 2025

This thesis is submitted for the degree of *Doctor of Philosophy*

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Virginia Claire Blessing
May 2025

Abstract

End-to-end encryption (E2EE) is one of the most significant improvements to end-user privacy in the last decade. The core principle of E2EE is that encrypted data can be decrypted only by the client devices at each end of the communication. In particular, data protected using E2EE cannot be decrypted by third-party service providers even under threat of legal mandate.

While instant messaging applications such as WhatsApp are the most widespread use of E2EE, deployment is gradually spreading to cloud storage, authentication credentials, email, and other services. Prior work has studied the underlying E2EE protocols in great depth, but building an E2EE system that is both secure and usable by the general public requires far more than a robust protocol and implementation.

As an in-depth case study of the challenges involved in building an E2EE system, we begin by considering the prospect of messaging interoperability between E2EE services. We present specific open questions and challenges around enabling interoperable E2EE messaging, discuss where current solutions fall short, and explore possible mitigations. E2EE messaging interoperability was recently mandated in the European Union and raises two fundamental questions: how to enable the actual message exchange, and how to handle the numerous residual challenges arising from encrypted messages passing from one service provider to another—including but certainly not limited to content moderation, user authentication, key management, and metadata sharing between providers. While championed not just as an antitrust measure but as a means of providing a better experience for the end user, interoperability runs the risk of making both the level of security and the overall user experience worse if poorly executed.

Even the most robust E2EE protocol is only as strong as the security of the keys used. Most contemporary mobile devices offer hardware-backed storage for cryptographic keys and other credentials, protecting keys from extraction by an adversary who has compromised the main operating system, such as a malicious third-party app. We survey trusted hardware usage in Android apps and find that despite industry-wide initiatives to encourage adoption, just 5% of apps collecting some form of sensitive data use the strongest form of trusted hardware, a secure element distinct from the main processor. In order to better understand performance of key storage options, we run experiments on all

widely used Android devices and find notably slower runtimes in more advanced hardware storage mechanisms, a reality which app developers must take into account when weighing security and usability.

Finally, E2EE has brought both benefits and challenges for usable authentication and recovery. We systematize cross-device credential syncing protocols made possible by E2EE, with a particular focus on "passwordless" authentication. At the same time, given that the nature of E2EE requires that the provider cannot recover data for users who have forgotten passwords or lost devices, inadvertent loss of data protected by E2EE is a major concern. We survey authentication and recovery schemes across all widely-used E2EE web services and find that the risk of account loss has prompted providers to deploy authentication and recovery schemes that are both more diverse and more easily compromised than conventional password-based schemes.

Acknowledgements

First, I would like to thank my advisors, Ross Anderson and Alastair Beresford, for their guidance and encouragement. This dissertation would not have been possible without the support and intellectual freedom they provided. Ross, the time I was fortunate enough to spend working with you made me a better writer, researcher, and person. I am also deeply grateful to both Ross and Shireen for opening their home to me for many lovely dinners and afternoon teas during my first few years in Cambridge.

There are many collaborators, office mates, and friends who made these years not only productive but enjoyable: Nicholas, Daniel, Ceren, Alex, Luis, Michael, Ilia, Dimitrije, Jessica, and Laurie; everyone in Cybercrime, my first home in the lab: Tina, Jack, Gilberto, Kieron Ivy, Anna, Anh, Yanna, Ben, Daniel, Richard, and Alice; the Specter Lab at Georgia Tech, particularly Mike and G; and many others in KCGS. I'm grateful to all of you for both the camaraderie and the chaos.

This dissertation would not have been possible without the financial support of Entrust and Nokia Bell Labs. I would particularly like to thank Pali Surdhar and Khaled Baqer of Entrust, who were unfailingly supportive and encouraging throughout.

Lastly, thank you to my family for their unwavering support of all my endeavours during the past four years and the many smaller steps that led here. There are countless others over the years who were generous with their time and wisdom in ways large and small but are too numerous to name. Thank you.

Contents

1 Introduction **15**

1.1 Contributions 18

1.2 Publications 19

2 Background **21**

2.1 Overview of End-to-End Encryption 21

2.1.1 E2EE Protocols 22

2.1.2 E2EE Credentials 22

2.1.3 Move Towards E2EE Storage 23

2.1.4 E2EE Storage Deployments 24

2.2 Trusted Hardware 25

2.2.1 Software-Backed Key Storage 25

2.2.2 Trusted Execution Environment 26

2.2.3 Secure Element 26

2.2.4 Usage Considerations 27

2.2.5 Trusted Hardware Best Practices 28

2.2.5.1 Legal Mandates 28

2.2.5.2 Developer Guidelines 28

2.3 Contemporary Authentication and Recovery Schemes 29

2.3.1 Primary Authentication Mechanisms 29

2.3.1.1 Passwords 29

2.3.1.2 Password Managers 30

2.3.1.3 Single Sign-On 30

2.3.1.4 Decentralized Identities 31

2.3.2 Secondary Authentication Mechanisms 31

2.3.2.1 Email and SMS 2FA 31

2.3.2.2 Authenticator App 32

2.3.2.3 Biometric Authentication 33

2.3.2.4 Hardware Tokens 34

2.3.2.5 Recovery Questions 35

2.3.2.6 Risk-Based Authentication 35
2.3.3 Recovery Mechanisms 36
2.3.3.1 Social Authentication 36
2.3.3.2 Long-Term Recovery Key 40
2.3.3.3 Manual Recovery 41
2.3.3.4 Break-Glass Encryption 41
2.4 Summary 42

3 On Securing Interoperable End-to-End Encrypted Messaging 43
3.1 Implementation Paths 45
3.1.1 Standard Protocol 46
3.1.2 Client-side Bridges 47
3.1.2.1 Restrictions on API Access 48
3.2 Open Challenges 49
3.2.1 User Identity 49
3.2.1.1 Cryptographic Identity 49
3.2.1.2 Real-world Identity 50
3.2.2 Key Distribution 51
3.2.2.1 Key Transparency 52
3.2.3 User Discovery 53
3.2.3.1 Centralized Directory Service 55
3.2.3.2 SPIN 56
3.2.4 Spam and Abuse 57
3.2.4.1 Existing Techniques 57
3.2.4.2 Content Detection Schemes 59
3.2.5 User Interface Design 60
3.2.5.1 App Selection Interface 60
3.2.5.2 Communicating Changes in Security Guarantees 61
3.2.5.3 Blue Bubbles and Green Bubbles 62
3.3 Current Status 63
3.4 Summary 64

4 KeyDroid: A Large-Scale Analysis of Secure Key Storage in Android Apps 67
4.1 Key Storage in Android 69
4.1.1 Software-Backed Key Storage 69
4.1.2 Hardware-Backed Key Storage 70
4.1.2.1 Trusted Execution Environment 70
4.1.2.2 Secure Element 70

4.1.3 Related Work . . . 71
4.2 Methodology . . . 73
4.2.1 Dataset Selection . . . 73
4.2.2 Play Store Data Safety Labels . . . 73
4.2.2.1 Published Data Safety Information . . . 74
4.2.2.2 Developer Self-Reporting . . . 74
4.2.3 Static Analysis . . . 75
4.2.3.1 Inter-Procedural Call Graph Analysis . . . 75
4.2.3.2 Package Analysis . . . 76
4.2.3.3 Obfuscation . . . 77
4.2.3.4 Reachability . . . 77
4.2.4 Performance Measurements . . . 77
4.2.5 Limitations . . . 77
4.3 Secure Hardware Usage in Android . . . 78
4.3.1 Overall Usage . . . 80
4.3.1.1 StrongBox Usage . . . 81
4.3.2 Usage by Category . . . 82
4.3.3 First-Party vs. Third-Party Usage . . . 82
4.3.3.1 Third-Party Libraries . . . 83
4.3.4 Key Authentication . . . 83
4.3.5 Implementation Security . . . 85
4.3.5.1 Ciphers . . . 85
4.3.5.2 Defaults . . . 86
4.3.5.3 Randomized Encryption . . . 86
4.3.5.4 Key Attestation . . . 87
4.4 Key Storage Performance . . . 88
4.4.1 Performance Evolution . . . 89
4.4.1.1 Key Generation . . . 89
4.4.1.2 Encryption . . . 90
4.4.2 Performance vs. Payload Length . . . 91
4.4.3 Cross-Provider Performance . . . 92
4.5 Developer Survey . . . 94
4.6 Summary . . . 95

5 Keys Not Under Doormats: Recovery in End-to-End Encrypted Systems 97
5.1 Related Work . . . 98
5.2 E2EE Recovery Mechanisms . . . 99
5.2.1 Recovery Codes . . . 101
5.2.2 Human-Memorable PINs and Passcodes . . . 102

5.2.3 Manual Reset . . . 103
5.3 Takeaways from E2EE Recovery . . . 103
5.3.1 Usability Improvements . . . 103
5.3.1.1 Cross-Provider Syncing . . . 104
5.3.1.2 Distributing Trust . . . 105
5.4 Summary . . . 105

6 Protecting Authentication Credentials Using End-to-End Encryption 107
6.1 Authentication Credentials . . . 109
6.2 Device-Bound Credentials . . . 109
6.3 E2EE-Synced Credentials . . . 111
6.3.1 Cloud Backups vs. Routine Device Syncing . . . 111
6.3.1.1 Backups and recovery . . . 111
6.3.1.2 Syncing between known devices . . . 111
6.3.2 First-Party vs. Third-Party Credential Syncing. . . . 112
6.3.2.1 First-Party Credential Syncing . . . 112
6.3.2.2 Third-Party Credential Syncing . . . 112
6.3.3 Attacks on E2EE Cloud Credential Storage . . . 113
6.4 Passkeys . . . 116
6.4.1 Authentication . . . 116
6.4.2 Cross-Device Authentication . . . 116
6.4.3 Backup and Recovery . . . 117
6.4.4 Exchange Between Known Devices . . . 117
6.4.5 Passkey Usability . . . 118
6.4.5.1 Credential Sharing . . . 118
6.4.5.2 Credential Revocation . . . 118
6.4.5.3 Credential Use Cases . . . 118
6.4.6 Passkey Deployment and Availability . . . 119
6.4.7 Passkeys and E2EE systems . . . 120
6.5 Summary . . . 120

7 Conclusion and Future Work 123
7.1 Future Work . . . 124

Bibliography 127

8 Secure Key Storage 173
8.1 Android App Manual Analysis . . . 173
8.2 Android Developer Survey Questions . . . 174

Chapter 1

Introduction

End-to-end encryption (E2EE) is used by billions of individuals around the world each day. Many of these users do so without even realizing they are using encryption, as E2EE is typically automated from the user's perspective and is simply a subtle and unobtrusive feature integrated into an existing cloud service. For the purpose of this thesis, E2EE communication is broadly defined as message transports in which user data at rest is encrypted prior to leaving client devices such that only legitimate end-users can decrypt the data. The spread of E2EE has been an enormous feat of real-world security, privacy, and usability engineering. While E2EE is still primarily deployed in instant messaging applications such as WhatsApp, its use is growing in cloud storage and backup services, email applications, and other web services with each passing year.

The core principle of an E2EE service is that encrypted data cannot be accessed by the service provider. Cryptographic keys are typically stored on client devices at each end of the communication, or protected with a user-chosen master password in the case of web applications. As such, E2EE guards against a wide range of threats to user privacy ranging from insider employees and Internet Service Providers (ISPs) to governments. In addition to general user security and privacy benefits, E2EE is often of interest to companies because it reduces legal liability. A company needs to expend far fewer resources securing data collected responding to external data access requests if the company cannot read the data in the first place. The corollary to this lack of data access, however, is that companies are no longer able to retrieve user data when requested to do so by law enforcement, a reality that has set off years of debate over E2EE deployment [18, 237, 141, 299]. Most recently, in February 2025 Apple withdrew its E2EE cloud storage service from the U.K. after the U.K. government requested backdoor access [237], prompting a legal battle which is ongoing.

However, while E2EE protocols are capable of encrypting data such that it cannot be decrypted by even the most robust adversaries, deployed E2EE services are vulnerable to all of the same social and device-based attacks that have plagued non-E2EE services for

decades. In 2020, a plot to kidnap the governor of Michigan was foiled after an internal informant shared copies of E2EE chat messages and other key information with law enforcement [21]. In March 2025, the U.S. national security advisor accidentally added a journalist to a Signal group chat discussing sensitive military operations seemingly due to an automated but incorrect contact update suggestion [214]. Critically, both of these instances were failures of operational security rather than cryptographic security. 'Signalgate', as the debacle came to be known, was also noteworthy since it is not recommended for national security officials to use a personal mobile device due to the risk of iOS and Android exploits compromising Signal chats at the communication endpoints. Instant messaging apps and other communications systems are also vulnerable to metadata analysis even if the message contents are encrypted [199, 326].

This dissertation studies how E2EE systems can become vulnerable in ways unrelated to the E2EE protocol itself. Specifically, we look at three open problems in building E2EE services: interoperable E2EE messaging, cryptographic key storage, and account authentication and recovery when the provider cannot access a copy of the keys.

We begin by considering the prospect of interoperable E2EE messaging, a case study which highlights the numerous practical considerations in implementing E2EE communications. The European Union has mandated that as of March 2024, WhatsApp and Facebook Messenger (the two messaging platforms deemed large enough to qualify under EU regulation) are obligated to provide other messaging services with the opportunity for their respective users to exchange messages. There are numerous relevant design specifics here which we discuss in depth in Chapter 3, but messaging interoperability opens up a Pandora's box of security and privacy challenges, touching on every aspect of communications security from the key exchange protocol to interface design. In particular, interoperable messaging disrupts existing user mental models of E2EE messaging. From a user's perspective, it is not necessarily clear whether they are communicating with someone on the same service or a different service, what metadata is being sent where, and even whether they are talking to the correct user. Moreover, there are real differences in provided security levels of widely used E2EE messaging apps. One consequence of the focus on E2EE protocols to the detriment of other system design aspects is that communication services which are broadly "E2EE" often blend together and are perceived as being equivalent or interchangeable. Telegram in particular benefits from this melding in public perception despite the fact that one-to-one chats are not E2EE by default and group chats are never E2EE [126].

We argue that while it is possible to achieve interoperable end-to-end encrypted communications with contemporary messaging services, this will require numerous new protocols and processes, both cryptographic and human, to maintain reasonable levels of security and usability. The conceptual simplicity of messages passing back and forth

between services belies the difficulty of the problem. Interoperability doesn't just mean co-opting existing cryptographic protocols so that one service provider can pass messages along to another—it encompasses the many supporting features and protocols that make up contemporary E2EE applications. Our discussion focuses primarily on the supporting aspects of E2EE communication agnostic to the method of message exchange, including user identity and discovery, key distribution, and spam and abuse.

There are two specific design considerations in E2EE systems that can have critical implications for security: key storage and key loss. The security of any encrypted service (E2EE or otherwise) reduces to the security of the cryptographic keys used. An attacker who compromises relevant keys can easily impersonate one end of the communication or decrypt stored data, rendering the encryption useless. Trusted hardware is one of the most promising tools to increase the real level of security provided to smartphone users. Once encryption keys are generated and stored within the hardware, they cannot be extracted or otherwise compromised even in the face of physical device access. Chapter 4 presents real-world usage rates of trusted hardware in Android applications and accompanying performance. Runtime performance is often prohibitive: while hardware-backed key storage using a coprocessor is viable for most common cryptographic operations, secure elements capable of preventing more advanced attacks make performance infeasible for symmetric encryption with non-negligible payloads and any kind of asymmetric encryption.

For all its benefits to user privacy, E2EE brings serious downsides for usability: compared to regular web services, the nature of E2EE requires that the provider cannot recover data for users who have forgotten passwords or lost devices. More robust recovery schemes are therefore required, leading to a plethora of solutions ranging from randomly-generated recovery codes to social authentication. Chapter 5 presents a comprehensive survey of deployed authentication and recovery mechanisms for end-to-end encrypted data, where either the web service or credentials may be E2EE, and Chapter 6 analyzes the specific case study of E2EE-based cloud credential syncing. There are two primary requirements for general-purpose authentication credentials: (1) credentials must be syncable across multiple devices (including cloud backup) and (2) credentials must be readily accessible but not guessable. E2EE is critically relevant to both of these goals, but in ways nuanced and complex: for goal (1), E2EE provides vital benefits as it would not be possible to securely sync and back up credentials without E2EE. For goal (2), however, E2EE services undeniably complicate authentication—the risk of account loss has prompted providers to deploy authentication and recovery schemes that are both more diverse and more easily compromised. Every E2EE service must balance the value of data protected to the end-user and consequences of loss with ease of compromise by an external attacker. Protecting keys from malicious compromise while simultaneously storing keys long-term such that it is improbable a legitimate user will lose access is a challenging task that this

dissertation presents and seeks to address.

In this dissertation I argue that systems using E2EE protocols can be designed in ways that undermine protocol security. E2EE is not a binary property of a system, though it is often treated as such. Increasingly, E2EE forms a baseline for system security, with substantial variety among services in both protocols used and other system design aspects. This dissertation explores several specific challenges that have as yet gone under the radar, including interoperable identities, insecure key storage, and recovery schemes that are either overly strict or overly lax. The ultimate goal of this research is to ensure advanced security features such as E2EE continue to be usable by and worthwhile for the general public.

1.1 Contributions

In this dissertation we aim to improve the security of deployed E2EE systems through focusing on two main design aspects: key storage and key recovery. We provide developers with empirical analysis and recommendations to better understand what schemes are used by others and factors influencing these choices. We further provide objective guidance to policymakers and industry practitioners attempting to interoperate messaging services in an effort to preserve the existing security and privacy properties of E2EE communications systems.

In summary, the main contributions of this dissertation are:

- A systematization and analysis of open considerations in interoperable E2EE messaging, motivated by the largest policy mandate concerning E2EE messaging in the last five years (Chapter 3).
- *KeyDroid*, a publicly-available tool for static analysis of key storage in Android apps which detects usage of all trusted hardware libraries (Chapter 4).
- A large-scale static analysis of ∼500,000 apps to understand how trusted hardware is used, cross-referencing results with user data collection practices (Chapter 4).
- A developer survey to better understand factors influencing developer decisions about trusted hardware (Chapter 4).
- Comprehensive measurements of key storage performance on all hardware primitives in Android devices, which provide developers with concrete guidance on trusted hardware usage patterns and performance (Chapter 4).
- An analysis and discussion of currently deployed authentication and recovery schemes in E2EE web services, including discussion of concrete research directions in E2EE

authentication based on observed gaps between industry deployment and research literature (Chapter 5).

- An analysis of the security of E2EE cloud credential syncing, including important security distinctions between high-level syncing protocols and an in-depth discussion of passkey variations (Chapter 6).

1.2 Publications

Some chapters of this dissertation have already been published, while others are currently under review. We describe each publication included in the dissertation below:

1. Jenny Blessing and Ross J. Anderson. **One Protocol to Rule Them All? On Securing Interoperable Messaging**. In Proceedings of Security Protocols XXVIII: 28th International Workshop, 2023. [93]

   The messaging interoperability position paper forms the basis of Chapter 3. For this chapter I adapted all text to fit with the dissertation, and updated the original text to include the current status of industry deployments and conflicts. I performed all analysis and background research and was the primary author of the text. Ross contributed towards the development of the ideas and writing.

2. Jenny Blessing, Ross J. Anderson, and Alastair R. Beresford. **KeyDroid: A Large-Scale Analysis of Secure Key Storage in Android Apps**. Under submission. [96]

   The *KeyDroid* paper forms the basis of Chapter 4. I adapted the text and figures from the original paper to suit the dissertation. I performed all data collection and conducted the experiments and subsequent analysis, and was the primary author of the text. Ross and Alastair contributed towards the development of the ideas and their presentation.

3. Jenny Blessing, Daniel Hugenroth, Ross J. Anderson, and Alastair R. Beresford. **SoK: Web Authentication in the Age of End-to-End Encryption**. In Proceedings of Privacy Enhancing Technologies (PoPETS), 2025. [94]

   This systematization of knowledge paper forms the basis of Chapter 5, which discusses recovery options in E2EE services, and Chapter 6, which discusses how E2EE is used to securely store and sync credentials. For these chapters I adapted the text, figures, and plots to fit with the dissertation. I led the conceptualization, analysis, and was the primary author of the text. My co-author Daniel was the main author of the FIDO2 protocol analysis and developed the accompanying figures. Daniel, Ross, and Alastair contributed towards the development of the ideas and their presentation.

Chapter 2

Background

In this chapter, we introduce several important security design paradigms on which the remaining chapters build. Section 2.1 defines E2EE in greater depth and summarizes its current deployments, while Section 2.2 similarly presents common forms of trusted hardware. In Section 2.3, we conduct a comprehensive analysis of contemporary authentication and recovery schemes. In order to consider suitable authentication schemes for E2EE contexts specifically we must first survey all available authentication schemes to glean lessons for usability and security.

2.1 Overview of End-to-End Encryption

In the client-server architecture pattern, user data frequently passes through third-party servers, such as a messaging application's web server. In this context, "third-party" refers to all parties other than the intended end-users and considers the service provider itself to be a third party. The core principle of E2EE is that encrypted data can be decrypted only by the ends of the communication (i.e., client devices). E2EE requires that any data is encrypted prior to leaving the end-user's client device using decryption keys that never leave the other client device. Critically, the service provider is unable to decrypt the data under any circumstances, even under threat of legal mandate, as the decryption keys are stored locally. From a business perspective, the goal of E2EE is to combat insider threats, including provider employees and the compromise of provider infrastructure by a malicious third party. A further consequence of E2EE is to render ineffective government warrants requesting data access as the providers cannot turn over data they cannot themselves access [424].

Traditionally mentioned in the context of secure ephemeral messaging (e.g., WhatsApp), E2EE can be deployed for many scenarios where a user wants to preserve the confidentiality of their data against the service provider storing the data (though the provider will still retain some amount of metadata). Most widely used password managers have deployed

E2EE for well over a decade [279], and conference calling platforms such as Zoom and Microsoft Teams began rolling out E2EE in 2020 and 2021 respectively [225, 390], albeit only for premium users in the case of Microsoft Teams [316] and not enabled by default in either Zoom or Teams [316, 225].

Increasingly, E2EE is being deployed to secure long-term data storage through cloud-based services such as email and device backups in addition to ephemeral communications. This marks an important shift, as E2EE has historically been limited to securing ephemeral communications, or particularly sensitive data such as passwords. We include an in-depth discussion of E2EE cloud storage below as this is a comparatively recent development within industry and these accounts tend to be of high value for end users.

2.1.1 E2EE Protocols

The core of any E2EE system is the underlying cryptographic protocol which enables the E2EE key and message exchange. As of 2025 there are close to half a dozen E2EE protocols in widespread use, with substantial variety in user counts and degree of auditing. Signal, WhatsApp, Viber, Facebook Messenger, and others rely on some variation of the Signal protocol, though they have developed different implementations (and, particularly in the case of group communications, different protocol versions). The Signal protocol remains the only open-source E2EE protocol that has been deployed at a scale of billions of users. It has also undergone over a decade of scrutiny from the cryptography community and is widely regarded as the gold standard.

There are also several proprietary protocols in widespread use. Telegram, Threema, and iMessage use custom protocols for the end-to-end encrypted layer, with Threema even using a custom client-to-server protocol [421, 411]. The Matrix Foundation has developed the federated and interoperable Matrix protocol (with associated E2EE protocol Olm), used in Element and other Matrix clients [298]. Matrix, Threema, and Telegram have all been subject to high-profile vulnerabilities in recent years [348, 28].

2.1.2 E2EE Credentials

There are several types of credentials involved in building an E2EE system, the distinctions between which are important when discussing authentication and recovery:

1. **User-facing account login credentials**: Most authentication today is still done using passwords, passcodes, and other forms of knowledge-based authentication where the user provides a human-memorable string as evidence of their authorization. To the end-user, the day-to-day authentication process in E2EE systems is similar (and often even identical) to non-E2EE systems. The difference between E2EE and non-E2EE authentication lies in the lack of conventional account recovery mechanisms to which

users have become accustomed (namely, password reset in case of a forgotten or lost credential). Given that account recovery is inherently an uncommon circumstance, the helplessness of the provider is a distinction users sometimes fail to appreciate until it is too late (discussed in greater depth in Chapter 5).

2. **Cryptographically-generated account login credentials**: An increasingly common paradigm is to authenticate the user using something they possess, rather than something they know, which translates to cryptographic credentials invisible to the user. Biometric authentication and all forms of device-bound credentials fall under this category, meaning that should the user lose the device or ability to biometrically authenticate, they have no recourse to recover access if that was their only authentication scheme. A key focus of this work is discussing mitigation strategies around device-bound credentials in particular.

3. **E2EE protocol credentials**: E2EE protocols involve numerous public/private key pairs used to enable secure key exchange and encrypt and decrypt content, including long-term identity key pairs, short-term session keys, and pre-shared keys. In this work, we abstract these details to focus only on the long-term private keys no longer stored with the service provider.

2.1.3 Move Towards E2EE Storage

Historically, cloud service providers have opted not to encrypt account data storage end-to-end due to political pressure [18, 17] and concerns over usability and account lock-out [226]. Apple in particular has come under pressure in the past from the U.S. government over encrypted cloud storage specifically: around 2017, Apple abandoned internal plans to encrypt cloud storage backups after the FBI objected [236]. From a usability standpoint, academic studies on the usability of multi-factor authentication apps have found account lockout to be a repeated concern for users since app backups are generally encrypted [181, 367]. The risk that users may lose long-term data is likely part of the reason that E2EE storage has been slow to catch on even among providers who deployed E2EE for messaging several years prior. WhatsApp, for instance, rolled out E2EE communications in 2016, but did not offer E2EE backups of these communications until 2021 [314].

In the wake of a changing technical and political landscape, however, Apple has publicly advocated for protecting cloud data with end-to-end encryption [226] and rolled out an opt-in E2EE backup scheme in late 2022). When it was revealed in February 2025 that the U.K. government had served Apple with a legal notice to provide alleged backdoor access to encrypted iCloud storage, Apple responded by removing the feature altogether rather than comply (and so U.K. users can no longer benefit from comprehensive E2EE backups) [237].

As of March 2025, Apple is locked in a legal battle with the U.K. government over its refusal to provide access to encrypted backups [364] and has received public expressions of support from lawmakers and national security officials in the U.S. government concerned about the security implications of weakening encrypted systems [238].

Apple's public justification for making E2EE backup opt-in for users, rather than the default, is to reduce the risk of permanent data loss as "the feature requires the user to take ultimate responsibility for managing their cryptographic keys" [226]. While Apple's deployment represents a major step forward for cloud data privacy, encrypted data storage for the general public requires us to revisit what security-usability trade-offs are acceptable since a provider by definition cannot restore access. A well-designed user interface can warn the user of the consequences if they lose their recovery credentials, but this could also serve to deter users from opting in. While Apple has declined to report statistics on what percentage of users opted to enable E2EE cloud backups, it is likely to be minimal due in part to concerns over recovery. To encourage widespread adoption of E2EE storage, we need schemes which are suitable for the average user.

Encrypted data storage is similar to non-custodial wallets in the cryptocurrency ecosystem, a space notorious for tales of irreversible data loss. Non-custodial wallets require the user to control their own private keys, instead of having the keys managed by their service provider [147]. The corollary to this setup is that the user has no recourse if they lose their private key, leading to permanent loss of the currency stored in the wallet. Cryptocurrency firms have been grappling with this problem for years, though the problem setup is slightly different in that cryptocurrency storage is often accessed infrequently, while messaging and storage applications can be accessed multiple times per day (which can affect which schemes are considered viable for a general userbase).

2.1.4 E2EE Storage Deployments

Apple iCloud, Facebook Messenger, and WhatsApp have all deployed opt-in end-to-end encrypted backup services in the past two years. Since 2022, Apple's Advanced Data Protection scheme provides users with the option to encrypt their iCloud backups end-to-end, such that the encryption keys are replicated across all user devices. Previously, it had only been possible to encrypt most iCloud data such that Apple retained access. While several other services already offered E2EE storage, these services are largely targeted at a more tech-savvy userbase and have not seen mass adoption.

Specific protocols vary, but cloud storage providers generally achieve E2EE backup by storing decryption keys in hardware security modules (HSMs) such that the user authenticates to the HSM and the provider relays messages between the client device and HSM but has no access itself. The HSM is essential to guard the key material against physical attacks and to enforce rate-limiting of password guesses.

2.2 Trusted Hardware

While end-to-end encryption is arguably the software-based paradigm with the greatest privacy impact in recent years, novel processor designs offering trusted hardware have revolutionized the privacy of data at rest. Consumer device key storage has seen major security improvements over the past decade: almost all modern mobile handsets now offer some form of hardware-backed credential storage capable of protecting keys against an adversary with root permissions [307, 213].

At a systems level, operating systems provide three options for storing cryptographic keys and other sensitive credentials: a software keystore via long-standing credential management libraries that have generally been available for several decades, hardware-backed key storage logically separated from the main operating system (OS) to protect against OS compromise, and hardware-backed key storage located on a separate processor to guard against the most advanced logical and physical attacks. We discuss security properties and limitations of each below.

The defining feature of hardware-backed key storage is that keys are stored and used in hardware separate from the main operating system. All cryptographic operations using these keys take place within the hardware component. A compromise of the device OS, then, will not compromise any cryptographic keys or other processes running inside the hardware element. Provided the secure hardware is not compromised, these operations cannot be inspected or interfered with by the Android OS (e.g., an attacker who compromises the device cannot extract keys or use them to decrypt data stored off-device). Importantly, secure hardware ensures that keys will never be revealed in memory while they are used (and therefore cannot be accessed even by a privileged user).

In this chapter and subsequent results chapters, we focus particularly on the hardware capabilities of mobile devices. We use the terms "secure hardware" and "trusted hardware" interchangeably throughout this thesis to broadly characterize hardware-backed key storage.

2.2.1 Software-Backed Key Storage

Mobile devices have historically relied on software keystores which operate within the mobile OS and use the device's internal storage, where the keystore master key is typically derived from a user's lock screen passcode. Software key storage implementations are vulnerable to memory extraction attacks, where an adversary with root permissions in the mobile operating system can observe the key as it is decrypted in RAM while being used [285, 198]. Mobile applications are particularly vulnerable to such attacks since apps are long-running processes and keys stored in memory are not garbage collected until a process has terminated. Malware, malicious third-party apps, and other privileged users are all capable of compromising the underlying OS, including kernel access control

measures, and launching an attack of this sort.

2.2.2 Trusted Execution Environment

The most common form of hardware-backed storage (commonly called "trusted hardware" or "secure hardware") is the trusted execution environment (TEE). A TEE is a special mode of operation by the main processor (e.g., Arm TrustZone or Intel VT) intended to provide a more secure, logically isolated execution environment. It has its own operating system and is typically located on the main processor, which is divided into main OS and the TEE OS, also commonly referred to as the *normal world* and the *secure world*. The hardware used to protect the secure world from the normal world depends on the processor architecture: TrustZone is used for ARM-based systems and provides dual execution environments [355] while Intel x86 uses virtualization technology to provide similar support [221].

The primary security benefit of a TEE is to guard against kernel compromise, including malicious applications installed on the device which could request root permissions [307, 372]. The device kernel and applications run in the normal world, while the secure world (i.e., the trusted hardware) stores long-term cryptographic key material and performs operations using these keys. Trusted hardware has numerous other benefits for mobile device security, such as enabling hardware root of trust schemes to authenticate firmware running on the device, but in this thesis we focus on the direct security benefits to app developers for storing and using cryptographic keys.

While TEEs offer substantial benefits over software-backed key storage, including protection from memory extraction, they are still vulnerable to various physical attacks, including side-channel attacks. There have been several documented attacks on Intel SGX [426, 117, 427], which is an example of a TEE; most of these were side-channel attacks [330]. Prior work has also discovered several architectural design flaws in ARM TrustZone implementations leaving data stored even in TEEs potentially vulnerable to sophisticated threat actors [386, 110]. To protect against the most advanced attacks, a device needs to contain a hardware element entirely separate from the main processor: a secure element.

2.2.3 Secure Element

Most premium devices contain a secure element (SE), a form of hardware security module (HSM) which must have its own CPU, memory, storage, tamper-resistant packaging, and a true random number generator [58, 50]. While a TEE is a separate OS on the main processor, an SE is an entirely separate processor. An SE provides all the benefits of a TEE and more: the increased isolation from the main device OS and processor provides

resistance to various side-channel attacks, including cold-boot memory attacks and shared-resource attacks [58]. As with a TEE, cryptographic keys are generated and stored within the confines of the SE, and any operations performed using the key material take place within the hardware so the key never enters an application's host memory. Different hardware elements are not mutually exclusive—for instance, a mobile device containing an entirely separate SE will almost certainly also contain a TEE as part of its main processor. While secure elements are still comparatively novel in mobile handsets, Hugenroth et al. [213] estimated secure element availability in contemporary mobile devices and found that as of 2023, 96% of iPhones and 45% of Android devices offer some form of SE. We expect these percentages will increase in future years as older devices are cycled out.

Spurred on by the advanced security properties SEs can provide, industry firms have invested significant resources into encouraging the development and adoption of HSM schemes: Google launched the Android Ready SE Alliance in 2021, a "collaboration between Google and Secure Element (SE) vendors" that aims to make discrete hardware-backed storage (e.g., StrongBox) "the lowest common denominator for the Android ecosystem" and to facilitate interoperability and consistency across secure element vendors within the Android ecosystem [1, 206]. We discuss recommended best practices and legal mandates in further detail in Section 2.2.5.

Despite the industry shift towards SEs as the desirable and intended outcome, to the best of our knowledge there have been no prior studies on the usage or performance of this form of trusted hardware. This is of particular concern since SEs are widely acknowledged to reduce performance (an observation which we quantify and present in Chapter 4).

2.2.4 Usage Considerations

Trusted hardware is not a panacea: although hardware-backed key storage prevents keys from being exported off-device or revealed in memory, the keys can still be *used* on-device by an attacker with root privileges, a "fundamental limitation" of hardware-backed storage [124, 307]. Even so, the adversary will only be able to decrypt data stored on the device which, depending on the application, may limit the damage they can cause if they are unable to use the keys to decrypt data stored *off* the device (e.g., data stored on a remote server). Additional authentication requirements prior to key use can also substantially mitigate this risk.

Furthermore, the use of hardware-backed protection for cryptographic key material is "best effort" in the sense that the Android Keystore API uses the TEE if it is available on the device (or SE if specially requested), but reverts to a software-backed keystore otherwise. The default reversion to a software-backed keystore instead of throwing an error reflects a desire to support backwards compatibility and a fragmented Android ecosystem containing many different device vendors with different price budgets and

hardware specifications. Developers who desire to require hardware-backed storage as the minimum security level of their product can add runtime conditional checks hardware availability and adjust accordingly. A key goal of this dissertation, then, is to explore whether app developers do indeed request to use trusted hardware on devices where it is available.

2.2.5 Trusted Hardware Best Practices

2.2.5.1 Legal Mandates

In certain industries, developers have to abide by a heavy patchwork of regulatory standards governing data collection and processing. In the U.S. (the region in which our application dataset and ranking information are collected) since 1996 the medical sector has been governed by the Health Insurance Portability and Accountability Act (HIPAA)'s Security Rule [341], which specifies minimum security standards that health service providers must meet. The financial sector has a variety of SEC regulations and longstanding laws they must comply with, such as the global Payment Card Industry Data Security Standard (PCI DSS) that governs processing and storage of credit card data.

Regulatory standards in the financial and medical industries generally require that data is encrypted at rest. Specific implementations, such as use of secure hardware to store credentials, are usually not mandated directly. For instance, the American Medical Association acknowledges that since security is an "evolving target, and so HIPAA's security requirements are not linked to specific technologies or products" [77]. Rather, regulation often encourages adoption indirectly, such as a law that mandates a security standard only provided by the hardware-backed storage mechanism. For instance, an industry may be required to use a FIPS-compliant random number generator [340], which on a particular mobile device is only available via the HSM API. Moreover, even in cases where regulations provide little specific guidance, providers operating in heavily-regulated sectors are generally motivated to prioritize security within their product to keep pace with the sector in which they operate and minimize the risk that they could be charged with running afoul of the law.

2.2.5.2 Developer Guidelines

Android's published security guidelines for developers [45] recommends developers use the Android Keystore for long-term or multi-use keys, and the OWASP Mobile Application Security Testing Guide [5] recommends that developers "should always rely on" available secure hardware to store and use encryption keys. To audit app security, Android's "app security improvement program" [53] further scans all applications for various potential security issues upon initial submission and subsequent updates, including well-known

vulnerabilities (e.g., Logjam), unsafe encryption modes, and insecure connection issues. We are not aware of any analysis of key storage.

2.3 Contemporary Authentication and Recovery Schemes

We broadly group contemporary authentication and recovery schemes into one of three categories, *primary*, *secondary*, and *recovery*, based on the context in which each is most widely deployed. Primary authentication mechanisms are the first (and often the only) step required for identity verification, while secondary authentication mechanisms are usually only triggered after the primary authentication process finishes correctly. Both primary and secondary authentication mechanisms are not mutually exclusive in real deployments–providers often allow users to choose among multiple secondary authentication mechanisms, or use one secondary factor as a fallback for another (e.g., using recovery codes in case of MFA app loss).

Recovery authentication mechanisms are by definition intended to be rarely used and may justifiably require more effort or hassle on the part of the user. We devote particular discussion to schemes relevant in E2EE recovery, such as social authentication and break-glass encryption.

2.3.1 Primary Authentication Mechanisms

2.3.1.1 Passwords

Security and usability issues with passwords are legion [172, 171, 208, 408, 332, 99]. Decades of academic research has shown that users constantly forget passwords [101, 395], use guessable passwords [434, 98, 346], reuse passwords across different accounts and providers as a coping mechanism [127, 440, 14, 384, 220, 173], and opt not to change their password even when notified of password reuse or insecurity [186, 435]. In the contemporary threat landscape, however, even widely recommended security practices such as increasing password strength would do little to protect against phishing attacks [409, 172] and large-scale data breaches [290, 317] even if users were to adopt best practices. These concerns impact users at all skill levels—academic work has found the relationship between technical expertise and vulnerability to common attacks (including susceptibility to phishing attacks, password reuse, and choosing stronger passwords) is largely inconclusive [441, 264].

In addition to public databases allowing users to check whether their credentials have been compromised [3, 4], industry has deployed various cryptographic techniques to automatically alert users of password reuse and breach, such as Meta's Private Data Lookup (PDL) tool using private set intersection to check whether a user's password is contained within a server-side set of passwords exposed in data breaches [203]. Unfortunately,

academic work has repeatedly shown that the effectiveness of user notifications is limited: Only around a quarter of warnings resulted in users changing their password [409, 186]. Given the unavoidable tensions between security and usability in any password-based authentication scheme, passwords are increasingly viewed as a “legacy authentication mechanism” [204].

2.3.1.2 Password Managers

Password managers are a critical mitigation strategy to make it easier for users to handle vast quantities of credentials. While the technical community favors password managers for security reasons as they allow users to opt for higher-entropy passwords and eliminates the need to memorize credentials, academic work has shown that users’ primary motivation for use is convenience rather than security [34], with some users even consciously avoiding storing credentials for high-value accounts in a password manager even as they use it for credentials for less sensitive accounts [34]. Users’ thought processes when choosing which password manager to use also tends to be driven by financial cost (e.g., if one requires a subscription fee) rather than security [325]. In practice, users frequently do not use password managers to their maximal security advantage and largely use them to autofill low-entropy passwords [34]. Moreover, users are generally still required to remember a password for the password manager itself, or else the password to a third-party email service that can be used to authenticate to the credential manager [419].

2.3.1.3 Single Sign-On

Single sign-on (SSO) is a federated login technique that centralizes the responsibility for authenticating users with a single primary provider (most commonly Google or Apple [319]) using access delegation protocols such as OAuth and OpenID Connect. SSO adoption has been limited by both legitimate privacy considerations over data sharing with big tech companies [402, 82, 134, 319] and holdouts in adoption due to lack of trust in the underlying technology [401, 404], with prior work showing users are less likely to use SSO for more sensitive accounts [118]. The crux of the privacy issue is that the centralized provider (e.g., Google) will be able to observe all authentication attempts for a particular user, though there have been several promising academic proposals to reduce data sharing [243, 320, 197, 162]. Some platforms (such as GitHub) opted not to offer federated log-in to maintain greater control over the authentication process for their website [209].

There has also been a large body of academic work showing security vulnerabilities both in the underlying protocol [391, 403, 437, 286, 286] and deployed implementations [462, 400, 80, 180, 179, 444], including real-world cybercrime networks that maintain honeypot websites and collect OAuth access tokens [158]. Apart from specific security and privacy

concerns, single sign-on schemes inherently present a single point of failure [207, 100] and hence an attractive target for attackers.

2.3.1.4 Decentralized Identities

In a real-world context, government-issued identity documents form the primary authentication scheme for most people. Despite their importance for all aspects of life, even though these documents are sometimes lost or misplaced, to the point where the U.S. government has a website devoted to replacing lost or stolen identification documents (including birth certificate and social security card) [423] given the frequency with which this occurs.

Given the resilience and replaceability of real-world identity documents, a common proposal in academic literature and government policy is to digitize these documents and allow individuals to authenticate to web services (E2EE and non-E2EE) using their real-world identity [382]. One such scheme electronic identity scheme, eIDAS, has been deployed in the UK and European Union for several years [219]. In the US, this has become more widely discussed as some states have passed age verification laws requiring users to verify they are above a certain age prior to accessing a service [23]. The FIDO2 Credential Exchange Protocol discussed in § 6.4.4 is not specific to passkeys and could conceivably be repurposed to enable digital identity authentication, though this will require all to trust a third-party credential issuer service to convert real-world eID documents into some form of cryptographic identity. In addition to general privacy considerations and the ease of compromising paper documents (e.g., an adversarial relative or roommate), another obstacle to deployment of these types of schemes is the lack of widespread eID ownership among the general public in some regions.

2.3.2 Secondary Authentication Mechanisms

Common second factor authentication (2FA) mechanisms used historically and still today include recovery questions, email and SMS-based MFA, 2FA authenticator apps, hardware tokens, and risk-based authentication (e.g., using browser metadata to flag suspicious login attempts).

2.3.2.1 Email and SMS 2FA

Email and SMS 2FA are still widely used for recovery today [278, 107], both as the primary 2FA mechanism and as fallback authentication strategies for more secure 2FA schemes (such as backups of 2FA authenticator apps [181]). We consider email and SMS 2FA jointly as academic work has shown them to be roughly equally usable in terms of recovery success rates and user perception, since in both cases the recovery process requires users to authenticate to a frequently accessed account (either email or mobile device) and

enter a short code [266, 101]. SMS has been the most widely deployed 2FA mechanism for at least a decade [323, 30, 177], the most common form of which is a time-based one-time password (TOTP). Both code-based and link-based 2FA are vulnerable to social engineering ("real-time phishing"), as users can often be tricked into sharing the code even when told not to do so by the companies. SMS-based authentication codes have well-documented security issues and are easily intercepted via SIM swapping attacks and attacks on the SS7 protocol [388, 273, 239, 272, 292], but there are nonetheless certain scenarios (e.g., low value accounts, a user posses only a shared email account) that may make SMS 2FA more suitable for an individual use case. SMS 2FA also typically reveals the code on the device lock screen, and in email 2FA some prominent websites (including Google and Facebook) would reveal the code in the email header or preheader [26].

While end-users have historically considered email and SMS to be the most usable recovery options, it is nonetheless plausible that users may lose access to one of these factors—for instance, a user may list their university or corporate email as the recovery email for their primary personal email account, and later leave the university or company. Google reported in 2018 that 10% of users fail email or SMS 2FA [317] (e.g., if they no longer have access to the recovery email), though providers attempt to mitigate this by regularly reminding users which recovery options they have set.

2.3.2.2 Authenticator App

Mobile authenticator apps (e.g., Duo Mobile, Microsoft Authenticator) have seen low adoption among the general public [317, 187, 352] but are frequently mandated in university settings [13, 368] or by a small number of security-sensitive providers (e.g., Github [231]). Academic usability research has found that MFA apps are generally considered easy to use [122, 13, 137] but that users perceive the extra step required by MFA as a nuisance [129, 296, 368, 13, 133]. MFA apps are widely supported among the top domains [362].

Recovery from App Loss: After several years of widespread 2FA app deployment, the academic community has begun to investigate the consequences of app loss (which can frequently occur as a result of device reset, loss, or theft, or because the app backup not including as part of larger device cloud backup).

User concerns over device loss have been a frequent theme with app-based 2FA. Every widely used 2FA app offers different backup and recovery options [305]. Several offer cloud backups with iCloud, Google Drive, and sometimes other cloud services, and encrypts backups with a user-chosen password. Others use "backup codes" which the user is responsible for storing, and which are easily lost, or offer users the option of backing up to another device via QR code [181].

Prior work has found that recovery is a weak point for 2FA apps that undermines the

security mobile app authentication is intended to provide by resorting to the usual array of fallback authentication mechanisms: SMS, email, passwords, and manual recovery [181, 178]. A spate of recent work has focused on the consequences of losing access to a two-factor authentication (2FA) mechanism, a common scenario for users who use a mobile app for 2FA and a recipe for disaster as mobile devices are regularly lost, changed, damaged, or stolen. Gerlitz et al. [178] studied how service providers respond to users losing a 2FA mechanism. In 2023, they created accounts at 78 popular websites or mobile apps using 2FA to see what recovery procedures, if any, were described to the user. They conclude that not enough attention is given to recovery, with 28 of 78 services studied not mentioning anything during the setup phase what backup or recovery procedures, if any, might exist. Amft et al. [35] went a step further and analyzed real-world deployments of multi-factor authentication on 71 websites, contacting the sites through public email addresses, support forms as though they were a user who had lost their second factor and going through the manual account recovery process. They found significant variation and inconsistencies among sites, including discrepancies between a given site's documentation and actual procedures. Accounts on 10 of 71 sites were recovered simply by providing specific knowledge about the account that only the real account owner would know, while 13 sites required some form of official real-world identification, such as a government ID, to regain access. All in all, the authors identify 17 distinct recovery procedures and conclude that they "could not identify best practices regarding MFA recovery procedures" due to the variety.

Gilsenan et al. [181] studied backups of two-factor authentication apps that use time-based one-time passwords (e.g., a six-digit code that a user has 30 seconds to provide to a platform), finding commonly shared flaws in backup security, such as that the one-time password data is backed up in plaintext or that SMS is used to authenticate to the backup. The scope of all prior work in 2FA generally assumes two points: (1) that the user recalls or has access to the backup password and only the second factor is lost, and (2) that manual recovery is a possibility since the provider has the ability to reset or remove MFA.

2.3.2.3 Biometric Authentication

A commonly floated approach is for a user to provide some element of their real-world identity to the service provider, such as biometric data. While biometrics are certainly the most resilient form of user authentication, this form of data raises numerous questions around privacy and accuracy. As with recovery questions, biometrics are often used as a component of a multi-factor authentication process, such as unlocking a mobile device with Face ID when a user has also demonstrated physical possession of the device as in passkey authentication. However, with regular end-user hardware, we believe biometrics are not suitable as a standalone authentication factor for a cloud-based service. Consequently, we

do not consider biometric data to be a viable authentication path at scale.

In the last few years, new attempts have been undertaken to bridge the gap to allow identifying individuals on a global scale. One example is the Orb technology that is being deployed as part of the Worldcoin project [456]. The project deploys proprietary hardware and algorithms in order to achieve unique global identification of individuals with very low false-positive and false-negative rates. One use-case is to provide strong "proof of personhood" (and therefore Sybil resistance) for Web3 services even where users do not have access to other means identification such as government issued IDs.

2.3.2.4 Hardware Tokens

The well-known weaknesses of authentication flows that rely solely on passwords has motivated the adoption of 2FA technology. However, SMS-based 2FA (Section 2.3.2.1) remains vulnerable to SIM swapping and authenticator apps (Section 2.3.2.2) do not protect against live phishing attacks. Hardware tokens that perform interactive cryptographic protocols with the online service address these issues.

Today, most hardware-based authentication protocols are based on the specifications provided by the FIDO Alliance, which many large companies are part of. The Universal Second Factor (U2F) protocol [168] allows for interoperable hardware tokens that can provide an authentication proof to different web services. This additional factor is typically only requested when the user logs in for the first time on a new device.

When registering at a new web service, the client (e.g., the browser) provides the hardware token with origin information that includes the domain name. The hardware token then creates a fresh public-private key pair in its secure memory and derives a key handle based on the origin information. Both the public key and the key handle are then passed through the client to the web service.

For later authentication, the web service provides the key handle and a challenge to the client which passes it together with the origin information to the hardware token. The hardware token first verifies that the key handle and origin match. This prevents other web service from tricking the user to authenticating on a phishing website. In a second step, the hardware token signs the challenge with the stored private key which the website can verify using the public key.

Hardware tokens typically perform a simple test of user presence by requiring a simple button press on the device. This ensures that each authentication attempt is known to the user and a malicious app cannot perform authentication requests in the background. Web services can use the attestation keys provided by the hardware tokens to ensure that the user is using a device that fulfills certain certification standards. [168]

From a usability standpoint, academic work has shown repeated concerns over account lockout upon token loss [121, 367, 128, 157], in addition to general annoyance over the

hassle of having to carry and retrieve an additional physical component, which is often reflected in high login timeout or cancellation rates [121, 368]. Difficulty of account sharing is another significant concern with hardware tokens [367].

2.3.2.5 Recovery Questions

First conceived of in 1990 [465], recovery questions ask users to answer a series of questions based on personal knowledge of the account owner, where the questions are usually determined by the provider but sometimes also user-generated. Frequently used historically, recovery questions are now widely disgraced as a viable authentication scheme after numerous studies showing that answers are low-entropy and often guessable by other individuals personally close to the account owner [266, 379, 363, 241], a result that has likely only gotten worse as users share troves of personal data online [354, 255]. To make matters worse, studies have also repeatedly found that some users provide untruthful answers as a means of improving their account security [184, 101, 266], but which has a side effect of making it more likely that the user themselves forgets the correct answer. Bonneau et al. [101] concluded back in 2015 that it is "next to impossible to find secret questions that are both secure and memorable".

Usage-Based Questions: A suggested variation to reduce the guessability of recovery questions is to use "dynamic" recovery questions in which the answer changes based on account and/or device usage patterns [27, 200, 463, 202], as compared with the traditional "static" recovery questions described above. For instance, a service provider may ask questions based on geolocation data [22, 201]. Prior work found that users were particularly worried that they wouldn't be able to recall the correct answers and might lock themselves out of their own account [27] as certain types of data (particularly app installations) are easier to remember than others (e.g., SMS and call history) based on how frequently the data changes.

Authentication based on usage history raises particular concerns for at-risk demographics, such as older individuals who may struggle with memory recall, public figures whose location history is readily accessible, or domestic abuse victims where the attacker is generally familiar with location and communication history. In summary, we conclude that both static and dynamic recovery questions are unlikely to be suitable as either a standalone authentication mechanism or as a second factor for accounts.

2.3.2.6 Risk-Based Authentication

While not traditionally considered a second-factor since metadata is often unwittingly provided by the user, risk-based authentication (RBA) is a critical strategy providers deploy to distinguish legitimate logins from nefarious access [450, 176]. RBA broadly refers to

any technical strategy that protects against illicit account access but does not require any information from the user, most commonly asking users to verify their email address [107]. Providers have long resorted to using metadata indicators to determine whether a recovery attempt is legitimate, though high failure rates of 2FA (even basic email and SMS 2FA) [317] require companies to carefully balance which logins should be considered suspicious. Broadly, services use an immense variety of mitigation strategies, including throttling (limiting the number of guessing attempts), previously-seen IP addresses, geolocation data, or other metadata available from browser fingerprints [192, 317, 450, 176, 458, 136, 385], and sometimes even monitoring users' behavior post-login [195]. Typically, additional verification steps are only deployed if an authentication attempt is deemed suspicious based on the particular metrics and statistical framework used by the account provider.

Both Apple and Google rely heavily on the concept of trusted devices, where logged-in devices are sent a notification to confirm any additional access attempts flagged as suspicious [290, 317], though this is only relevant when multiple devices are connected to an account. Temporal lockouts, where the user needs to wait a certain period after a correct authentication, are commonly deployed in recovery schemes. For instance, as early as 2008 Gmail only allowed account recovery through recovery questions if the account in question had not been accessed in the past five days [138]. Today, Google deploys temporal lockouts as part of the manual recovery process to ensure a legitimate user has a chance to deny the request [190], though research has shown users often fail to recognize malicious sign-in attempts [295].

RBA is almost always deployed as an additional authentication measure on top of a pre-existing primary authentication scheme, most commonly in addition to standard password-based authentication [452, 451]. Prior work has shown it is possible to evade many RBA measures using various cloaking techniques [38, 338, 318, 339, 108, 281].

2.3.3 Recovery Mechanisms

2.3.3.1 Social Authentication

Social authentication and recovery, where an account holder designates one or more recovery contacts or "trustees" who can help them regain access, is the only recovery scenario that can dependably handle various real-world disaster cases (where the user loses or forgets all devices and passwords). First proposed by Brainard et al. [104] of RSA in 2006, social recovery takes advantage of real-world trust relationships to authenticate a user by having a different user vouch for them. Brainard et al. motivated this concept of vouching by considering the financial cost to the company of password-reset staff, but in today's world social recovery is an important option because of the difficulty of authenticating the correct user as part of a manual recovery process, and because in some

cases the provider may be unable to reauthorize the user (as in E2EE services), and has received renewed attention in recent cryptographic proposals [113].

Single Recovery Contact: The simplest social recovery scheme is to designate a single recovery contact (presumably a close acquaintance). LastPass, Bitwarden, and 1Password have all deployed some form of trusted contact recovery for a pre-designated user of the same password manager [268, 10, 89] (in Bitwarden only premium users are able to set an emergency access contact).

Apple's E2EE cloud backup scheme, introduced in 2022, also deploys the idea of a recovery contact, described by Apple as "a trusted friend or family member" [67]. In Apple's scheme, this contact is another iCloud user who generates a short code to send to an original user Alice enabling her to recover her account. In theory, a user with two iCloud accounts could also use one as the recovery contact for the other, though this arrangement would not adequately address the root concern. This is similar to a multi-wallet system in the cryptocurrency ecosystem, where a user leverages a secondary wallet to authenticate to the primary wallet [369]. Service providers can also allow a user to designate multiple recovery contacts, where each contact has the ability to single-handedly provide the account holder with a code to restore access.

While Apple's documentation [67] promises that a recovery contact "won't have any ability to access your account, only the ability to give you a code to help you recover your account", in practice a recovery contact could simply pretend to be the original user (assuming the contact knows a few additional basic facts, such as the iCloud account email used). Apple takes several precautions to prevent a recovery contact from gaining access to the account. Apple requires any individual entering the recovery code to answer an additional set of security questions in the first instance. Most importantly, Apple documentation suggests there is a time delay between the recovery request and regaining account access, and specifies that users should not use their devices in the intervening period because to do so would indicate that the request is not legitimate. In short, a user who is logged in and actively checking their devices and/or accounts will be able to detect that such a recovery process has been initiated. If the user has associated other traditional two-factor authentication mechanisms with the account, such as a phone number or email address with a different provider, they may also receive a notification on this platform informing them of the access request.

Threshold Social Recovery: As an alternative to designating individual users as an all-powerful recovery contact, service providers can also offer a threshold secret sharing scheme where the secret is divided among multiple recovery contacts but can be reassembled once a certain number of shares are combined [387]. In Shamir secret sharing, for instance, a key is divided into n pieces with a recovery threshold k such that k pieces (where k $\leq$

n) are needed to reassemble a valid secret. This has the benefit of making the recovery mechanism more resilient against unavailable or malicious recovery contacts by distributing trust among multiple contacts and providing redundancy in social recovery.

Schechter et al. [379] first proposed an account recovery scheme with multiple recovery contacts in 2009, where a key is split among the multiple contacts such that a minimum threshold of the total must share an account recovery code with the original user for them to regain access. Although Schechter et al. do not use the term Shamir secret and do not specify how the secret is divided up among the trustees, the scheme described functions in a similar manner to a standard cryptographic secret sharing scheme. Follow-on work has shown threshold trusted contact schemes are highly usable, with failures of trusted contact authentication due to time delays or timeouts (likely due to the perceived low value of the accounts used in experiments), rather than poor mental models or misconceptions [266, 379, 394].

Industry has already deployed secret-sharing schemes for recovering E2EE data. PreVeil, a cross-platform cloud service offering end-to-end encrypted email and file storage for enterprises, deployed an opt-in threshold scheme in 2019, stating that a system that distributes trust among a set of trustees is more secure than one which centralizes trust in a single trustee [359]. In PreVeil's design, when an account holder has initiated the recovery process, they will be presented with a list of their previously designated recovery contacts and able to select which members of the list should be used to approve the request. PreVeil's system architecture is somewhat unusual in that it does not use passwords or require the user to enter any credentials in order to log in. Instead, they store the user's private key on-device, allowing anyone authenticated to the device to access PreVeil storage. As a result, PreVeil needed a sufficiently failsafe backup mechanism in case the user loses their device(s) since no password or recovery code exists. Facebook used to offer a "Trusted Contacts" feature for regaining access to Facebook accounts (albeit not in an E2EE scenario) via a three-of-five secret sharing scheme [155], but deprecated the feature in 2022.

Social recovery has been gaining favor in the cryptocurrency ecosystem as well. In 2022 BitKey, a non-custodial hardware wallet (i.e., users store their own private keys), enabled an opt-in threshold social recovery scheme [89], where an account holder designates three recovery contacts and two of the three are needed to restore access. PreVeil does not require a certain threshold size, but similarly specifies in their documentation that two-of-three is a typical setup.

Limitations of Social Recovery: Social recovery goes a long way towards mitigating the challenge of an individual user managing their own keys, but at the same time presents several new concerns. While Apple takes several sensible precautions to prevent a recovery contact from gaining illicit access (instituting a time delay, requiring the individual

requesting access to answer a series of additional security questions, etc.), this mode of recovery is nonetheless vulnerable in certain scenarios. We can reasonably assume that someone close enough to the account holder to be designated a recovery contact will likely be able to answer any additional verification information (including the email address associated with the iCloud account, date of birth, etc.), and therefore gain access to the account.

A time-delay between the access request and when access is granted, during which the account holder is notified that a request has occurred, is essential for mitigating illegitimate access. However, time-delay schemes rely heavily on users' attentiveness and assume that users check an account regularly, and recent academic work found that users often fail to recognize and respond to login attempt notifications [295]. Critically, the dependence on this proactive detection on the part of the user means that security guarantees of social recovery do not hold if the account owner has passed away. This is not a scenario most users contemplate for obvious reasons, but posthumous account access is fairly simple under Apple's individual recovery contact scheme. [1] In the case of a single recovery contact we must also consider an honest-but-curious recovery contact, such as a relative who initially requests access to recover family photos but later realizes the account also contains years of messaging history. A threshold secret sharing scheme would partly mitigate this scenario since multiple contacts would need to agree that access is acceptable.

Social recovery has also been exploited by online scammers. The process of receiving an unsolicited message from an acquaintance asking the recipient to click on a link and provide some information closely resembles real-world scams, a fact which malicious actors used to their advantage. Facebook's Trusted Contacts feature was the target of a popular scam in 2017 in which an attacker who has compromised a given Facebook account sent messages to the account owner's contacts, pretending to be the owner and asking the recipient to click on a link to help them reset their password by providing the message sender with a recovery code [20, 229]. Unfortunately for the victim, the link provided was in fact a password reset link for the recipient's account, and the code the recipient sent back to the attacker allowed the attacker to compromise the recipient's account as well. Shortly after this scam became widely publicized Facebook disabled trusted contacts as a recovery mechanism, though the company never officially provided a reason.

A more contemporary concern is that social verification may be vulnerable to manipulation by generative AI tools, such as a falsified video call or voicemail, with even close contacts unable to distinguish between genuine and artificial content. Simply speaking to another user over the phone was considered sufficient identification as part of a social recovery scheme as recently as 2016 [394], but there have been numerous voice- and

[1] Apple has a separate notion of a "Legacy Contact", an optional setting where a user's legacy contact can recover account access by manually presenting a death certificate to Apple—but social recovery can intentionally or unintentionally also become a legacy contact.

video-cloning attacks in recent years used in real-world scams [111, 249, 15, 205]. Social authentication is all too easily susceptible to various social engineering attacks, such as where a contact calls from an unusual phone number and claims they have lost their smartphone and need assistance recovering an account—when in reality, the contact's voice is AI-generated.

2.3.3.2 Long-Term Recovery Key

There are several different terms for this concept ("recovery key", "recovery code", "master passphrase", etc.), but they all refer to a pseudorandom string that the user presents to the service to restore access to their account. This recovery code is generally distinct from a standard user-generated password or PIN regularly used to log in in that it is arbitrary and usually substantially longer to guard against brute-force attacks.

The popularity of recovery codes as a recovery mechanism endures because they are generally the most secure and efficient way of recovering account access—provided a user stores the code in a safe place and does not lose it. Virtually every cloud backup option either requires users to use a recovery key of some sort or offers it as an option if their protocol allows for multiple recovery mechanisms: Apple iCloud lets users use a 28-character recovery key (in addition to the ordinary iCloud password) [68], and WhatsApp encrypts backups with either a 64-digit encryption key or a user-generated password [448]. Meta's Labyrinth protocol offers several different recovery mechanisms, one of which is a standard 40-character recovery code [154]. Signal does not support cloud backup but lets users encrypt a local backup using a 30-digit recovery key which the user is responsible for storing and safeguarding. These codes are generally only shown once upon creation, although a logged-in user can usually also generate a replacement recovery code even if they have lost the old one.

A recovery key option suffers from the same problem as a user-generated password: users all too often forget or lose it. A small number of users may neglect to save it at all, often out of overconfidence that they will never need it [212]. Unintentional loss is even more likely given that the nature of a recovery key is that it is used rarely, if ever. A user may store it on local device storage or on a physical piece of paper, encounter it many months later, and throw it out without realizing its significance. If the user's only backup recovery mechanism is this passcode (in addition to losing access to their device and/or regular password), they have no recourse and are locked out of their account permanently. The simplest mitigation technique is to create redundant copies of the passcode, though this increases the attack surface and potentially requires users to store the passcode where family members or others can access it.

2.3.3.3 Manual Recovery

Manual recovery ("ad-hoc schemes" [212]) are the recovery scheme of last resort [266, 347, 175, 178]. Some large industry providers offer a formally described manual recovery process [189, 72], while most others offer generic support contact information. Providers may also offer appeals processes in cases where a provider's content moderation scheme flags the account [246]. On a large scale, however, there is little incentive for small service providers to expend significant effort of these types of schemes, especially for users of unpaid services. Importantly, provider-assisted recovery is inherently not possible in E2EE services, which usability research has shown that some users do not understand, with users of an E2EE email service mentioning provider assistance as a possible recourse after recovery code loss [212].

2.3.3.4 Break-Glass Encryption

To develop an E2EE variant of manual recovery, a recent thread of academic work has attempted to tackle challenges around encrypted data loss by focusing on detecting, rather than outright preventing, account access [378]. Orsini et al. [344] proposed a cryptographic scheme for emergency access to cloud data storage, using the same threat model as in this work where a user Alice has lost all relevant credentials and all devices. They propose a credential-less authentication scheme in which any user can request access to a cloud account knowing only the associated email address or similar username, but there are only two possible states for a given account: either the legitimate user Alice is logged in and can monitor and reject illegitimate access requests within a certain timeframe, or Alice has become locked out of her account (e.g., by losing her device) and her request to regain access will be automatically granted after some time period has elapsed (since there is no legitimate user to reject it).

Such schemes are entirely dependent on detectability: the assumption is that the legitimate user will be consistently online, and confidentiality is guaranteed by proactive action on the part of the account owner. Both the Orsini et al. scheme and a similar concept for cryptocurrency wallets [92] assume an information asymmetry between the legitimate user and all other users in that the legitimate user would know when they have lost access (and request to be restored to the account) before anyone else, but this does not always hold (e.g., a device is stolen, posthumous access, etc.). Perhaps the biggest concern with these "break-glass encryption" schemes is that a deceased or incapacitated account holder is now vulnerable to any relatives or acquaintances familiar with their account name to a far greater extent than was the case with existing social recovery schemes. We are skeptical that any such scheme would ever be feasible for the general public.

2.4 Summary

In this chapter we provided an overview of several innovative security design paradigms which have emerged in the past decade. We began with a summary of E2EE and its contemporary deployments, observing that E2EE has become increasingly widespread in applications designed to store data long term, as opposed to its historical use encrypting ephemeral communications data. This shift has been the subject of repeated and ongoing debate between industry firms, governments, and law enforcement since a deliberate consequence of E2EE is that the service provider can no longer provide as much raw data when served with a legal mandate.

Since cryptographic key security is essential to maintaining E2EE, in Section 2.2 we looked at contemporary key storage paradigms, with a particular focus on key storage in trusted hardware. We see that trusted hardware is widely available in mobile devices and comes in two general flavors: a secure area of the main processor, and a discrete secure chip. It is yet unknown, however, how widely secure hardware is used and how efficient different forms of hardware are in mobile devices. The background provided in this chapter sets the scene for empirical evidence of hardware usage and performance provided in Chapter 4.

Finally, the inability of the provider to access data encrypted using E2EE raises serious questions about which authentication and recovery schemes are currently in use in an E2EE application. In this chapter, we provide a high-level overview of all proposed and deployed authentication and recovery schemes in web applications (both E2EE and non-E2EE), sub-dividing authentication schemes into primary and secondary categories based on common usage. In particular, we discuss the prospect of social authentication in depth, a category of schemes which have flown largely under the radar in academic research and industry discussions but have increasingly been deployed in E2EE applications. We return to the subject and provide an analysis of which schemes are used in contemporary applications in Chapter 5.

Chapter 3

On Securing Interoperable End-to-End Encrypted Messaging

While much attention has been devoted to deployed E2EE protocols in both academic research and industry initiatives, E2EE systems are more commonly made insecure through system design choices unrelated to the E2EE protocol. E2EE messaging services come in a variety of interface designs and default privacy settings, at times leaving users vulnerable despite using a "secure" service. For instance, Signal removed interoperable SMS support in 2022 after repeated user misunderstandings over which messages were E2EE and which were SMS [331]. As we will discuss in this chapter, interoperable messaging touches on virtually every major design aspect of an E2EE messaging service, running the risk of exacerbating existing design challenges if poorly implemented.

Users of E2EE messaging services have long existed in a world where they need to use the same service as another user in order to communicate. A Signal user can only talk to other Signal users, an iMessage user can only use iMessage to communicate with other iPhone users, and so on. Platform interoperability promises to change this: the vision is that a user of a messaging service would be able to use their platform of choice to send a message to a user on a different service—following the precedent of email and SMS. Proponents of this kind of open communication have argued that it will benefit both the end user and the market for services. If users can message each other using their preferred service, they can enjoy their user experience of choice, and if there is less pressure to use a service simply because others use it, this can eliminate network effects and market monopolies.

An interoperability mandate for end-to-end encrypted messaging systems is no longer hypothetical: the European Union's Digital Markets Act (DMA) came into force in November 2022 with a compliance deadline of March 2024. Article 7 requires that the largest messaging platforms (termed "gatekeepers" by the DMA) allow users on smaller messaging platforms to communicate directly with users on the large platforms [149]. Ultimately,

the only two services deemed a gatekeeper are WhatsApp and Facebook Messenger, both owned by parent company Meta. In February 2024 the European Commission decided that while Apple's iMessage service technically met the requirements threshold, iMessage's market share in Europe was sufficiently small in comparison to WhatsApp and Facebook Messenger that it should not be considered a gatekeeper messaging service [357].

The mandate applies only to the gatekeepers (WhatsApp and Facebook Messenger), with any non-gatekeeper platforms free to choose whether they wish to interoperate with other platforms. In accordance with the DMA, the gatekeepers cannot deny any "reasonable" request. Notably, it leaves the technical implementation details for the platforms to determine. In the U.S., the 2021 ACCESS Act proposed similar requirements but has yet to make any headway in Congress.

In this chapter, we survey and explicitly articulate the security and privacy trade-offs inherent to any meaningful notion of messaging interoperability, focusing primarily on the supporting aspects of E2EE communication which are largely agnostic to the actual method of message exchange. Designing a system capable of securely encrypting and decrypting messages and associated data across different service providers raises many thorny questions and practical implementation compromises.[1]

We outline what current solutions exist and where existing protocols fall short, and propose high-level solutions for tackling some of these challenges. For the sake of the discussions that follow, we assume platforms make a genuine effort to develop a system that emphasizes security and usability, though in practice they may degrade the user experience for interoperability, whether as a matter of necessity (WhatsApp has more features than Signal) or choice (to maintain some degree of customer lock-in). But as we will discuss, these challenges exist even if platforms make real efforts to open up their systems.

The DMA includes a purported safeguard that the "level of security, including the end-to-end encryption, must be maintained" in an interoperable service, but this raises as many questions as it answers. "Level of security" goes well beyond the mere fact of using an end-to-end key exchange protocol. A platform may use a proprietary E2EE protocol that does not provide forward secrecy, a de facto standard in preventing compromise of past communications [102], does not regularly rotate encryption keys, or neglects various other cryptographic guarantees. Until recently, the Swiss messaging app Threema openly had no forward secrecy at the E2EE layer [410], a design which led to multiple security problems [348]. Will these be considered valid reasons for a gatekeeper to deny a request to interoperate?

[1] Messaging interoperability raises numerous systems challenges, such as the latency impact of transferring messages between service providers when users expect communication to be instantaneous, particularly with audio/video calls, impact of bridging on mobile device battery life, etc. We limit our scope to challenges with direct security impact.

How will a gatekeeper verify the requesting service's encryption protocol, along with the authentication and content moderation schemes used? Will they need to take the requesting service's word for it, or else invest the resources to do a proper security audit? If the requesting service is closed-source, will the gatekeeper have a right to request access to the other's source code should they wish to do an audit? All widely-used E2EE messaging services have significant differences in both protocol and implementation, including fundamentally different design decisions impacting security and usability [148]. For instance, WhatsApp and Signal, despite both being based on the Signal protocol, handle key changes when a message is in flight differently [223]. When a recipient's keys change in the course of a conversation (e.g., because they uninstalled the app), Signal discards any messages sent after the change, while WhatsApp chooses to deliver them once the recipient comes back online. Both designs are legitimate [416, 322].

The greatest challenges are non-technical: interoperability will require competing platforms to cooperate and communicate, both in the current design phase and after any agreed-upon interoperable scheme has been deployed. Each service provider will need to trust the others to provide authentic key material, enforce certain spam and abuse policies, and generally to develop secure software with minimal bugs. A vulnerability or outage in one service now propagates to all other services with which it interoperates. Existing cryptographic protocols mitigate, but do not eliminate, these trust requirements. There is simply no getting around the fact that interoperability represents a dramatic expansion in the degree of trust a user will need to place not only in their own messaging service but also in any used by their communication partners, a point to which we will return throughout the paper. Much of the rhetoric around messaging interoperability at a March 2023 European Commission-hosted stakeholder workshop [150] and elsewhere calls to mind the quip "if you think cryptography is your solution, you don't understand your problem."[2]

This chapter is based on the paper "One Protocol to Rule Them All? On Securing Interoperable Messaging" [93]. For this chapter I adapted all text to fit with the dissertation, and updated the original text to include the current status of industry deployments and conflicts. I performed all analysis and was the primary author of the text. Ross contributed towards the development of the ideas and writing.

3.1 Implementation Paths

There are two broad paths to enable separate messaging platforms to talk to each other [86, 302]: either all platforms adopt a common communications protocol (native

[2]This quotation has been alternately attributed to Roger Needham, Butler Lampson, and Peter Neumann [36, 182], and paraphrased by Phillip Rogaway [370].

interoperability), or each platform publishes an open API allowing others to communicate with them through a bridge. There are more flavors than these, including many hybrid possibilities with varying levels of implementation and maintenance feasibility—a platform could, for instance, support multiple protocols.

3.1.1 Standard Protocol

There are several existing candidates for selecting a universally adopted end-to-end key establishment protocol. The Matrix Foundation has developed the federated and interoperable Matrix protocol [298]. The Signal protocol (formerly TextSecure) has been around for roughly a decade now and is the only open-source E2EE protocol that has been deployed at a scale of billions of users. More recently, the IETF standardized a new E2EE message exchange protocol, MLS (Messaging Layer Security) [83], which is intended to provide more efficient group communications than in Signal and related protocols. In February 2023, the IETF created a new "More Instant Messaging Interoperability" (MIMI) working group, the IETF's latest messaging interoperability standardization effort, dedicated to establishing the "minimal set of mechanisms" needed to allow contemporary messaging services to interoperate [217]. Among other aspects of messaging standardization, MIMI will seek to to extend MLS to deal with user discovery ("the introduction problem") as well as content formats for data exchange [217]. Work is ongoing and expected to continue well into 2025.

Any choice here is not obvious. Several of the largest messaging services already use variations of the Signal protocol, and Meta explicitly advocated for widespread adoption of Signal at the European Commission's stakeholder workshop [150]. Another possibility is some sort of hybrid option: Matrix has proposed Matrix-over-MLS as part of the IETF's MIMI working group [414]. But any agreed-upon standard would eventually have to support many of the features and functionalities of all widely used E2EE applications.

Switching to a standard communications protocol poses immense challenges given the variety of protocols currently in use. Signal, WhatsApp, Viber, Facebook Messenger, and others rely on some variation of the Signal protocol, though they have developed different implementations (and, particularly in the case of group communications, different protocol versions). Telegram, Threema, and iMessage use custom protocols for the end-to-end encrypted layer, with Threema even using a custom client-to-server protocol [421, 411]. Element uses the Matrix protocol [298]. Existing messaging services would need to either switch to a common protocol or support multiple protocols. Service providers not only have to agree on a single protocol, but in many cases would have to redesign their entire system and manage a major migration to the new interoperable version.

There are also valid concerns around hampering innovation: Moxie Marlinspike, one of the co-creators of the Signal protocol, has argued that centralized, unfederated protocols

evolve more rapidly than decentralized ones, where the latter have to deal with a wide diversity of clients, implementations, and deployments [297, 321]. For these reasons and others, a common protocol seems less practical in the near future than client-side APIs, not least because of the time constraints imposed by the DMA.

3.1.2 Client-side Bridges

In the face of substantial political, economic and technical obstacles to a universal communication standard, the developer community has begun to gravitate towards the idea of providing interoperability via public APIs, at least in the short term [301]. Each service provider could largely keep their existing E2EE protocol and implementation, but would provide a client-side interface to allow other messaging services to interact via a bridge between the two services.

Depending on the precise architectural design, such an interface would entail decrypting messages locally on the recipient's device after they are received from the sender's service provider, and then re-encrypting them with the recipient service provider's protocol. If the bridge (and therefore the key establishment protocol) runs on the client, this does not break the general notion of end-to-end encryption, at least theoretically. Only the endpoints (the client devices) see the plaintext message; no platform server is able to decrypt messages at any point. While running a server-side bridge is a technical possibility, both Meta and Matrix have acknowledged that server-side bridging has been "largely dismissed" since it would require message decryption and re-encryption in view of the service (violating the core principle of end-to-end encryption) [303]. A hybrid of the two, in which only the E2EE protocol is run in a client-side bridge with a server-side bridge doing much of the actual message transport, is also a possibility to handle systems challenges that may arise from asking the client to do too much work.

Some service providers already offer similar interfaces, either to integrate with other services or to allow third-party clients access. iMessage, Apple's end-to-end encrypted messaging service which interoperates with SMS, provides an example of client-side bridging at scale. It is well worth noting, however, that the technical feasibility of the message exchange, even at large scale, does not mean that this can be smoothly applied to interoperating E2EE services. SMS suffers from serious privacy issues, as all messages are unencrypted, and interface design choices (namely, bubble color) have led to widespread aversion to SMS conversations. We will return to both points in greater depth later on.

Both of these approaches are far from a panacea. Eric Rescorla has written and spoken at length about the challenges of writing, standardizing, and implementing protocol specifications [146, 150]. Client-side bridging comes with its own set of challenges: each service requesting access will need to build a different bridge for each provider with whom they want to interoperate. This has inherent security implications arising from the amount

of excess code a service would need to add to achieve interoperability with a meaningful number of services. And while client-side bridging may not break E2EE in the technical sense that the plaintext messages are only viewable on the client's device, it is indisputably not a conventional meaning of E2EE. Moreover, providing *an* interface is not the same as providing a *usable* interface, which is a challenge even when a provider is giving their best effort. To aid in providing a reasonable developer experience, Rescorla has suggested a set of main interface requirements, including unambiguous interface specifications, stable interfaces, test servers, and real-time support from engineers at the other service [146]. Needless to say, these are far easier said than done.

3.1.2.1 Restrictions on API Access

In practice, a service provider's interface cannot truly be "open" due to the sensitivity of the data being exchanged. Large service providers will individually approve requests by smaller service providers. The process for filtering requests may vary by provider, though they cannot deny a "reasonable" request. An interoperable interface will likely use some sort of revocable access key to manage access. Providers then need to figure out how to manage key requests and key storage, including some sort of service-level identity indicator [222].

But API request filtering is not merely a gatekeeping measure: platforms need to be able to detect and block bulk spam and forwarding services in real time. WhatsApp relies heavily on behavioral features to detect spam clients at the time of account registration in its existing public-facing interfaces, stating that the majority of use cases of these interfaces are spam [300]. Will Cathcart, the current head of WhatsApp, identified the need to restrict or outright block certain services as one of the most important considerations under an interoperability mandate [109].

The EFF has raised the spectre of a malicious actor creating a fake messaging service with a number of fabricated users and requesting access [85]. Large service providers will need to set certain objective and justifiable thresholds that give them the flexibility to respond only to legitimate requests. This is seemingly within the bounds of the DMA, though of course this will depend on what types of requesting services the European Commission considers to endanger the "integrity" of a service [149].

Many of the data sharing and privacy concerns surrounding third-party access are not fundamentally new: Facebook's infamous Cambridge Analytica scandal was made possible through Facebook's external app API. What has changed with the DMA is that providers have far less flexibility to refuse or revoke access, let alone in real time as a situation is unfolding. Service providers will need a fair amount of latitude in their ability to deny access requests to continue to guard against malicious data scraping and mining, regardless of whether interoperable message is implemented through client-side bridging

or an open standard.

3.2 Open Challenges

The security and privacy considerations associated with trying to reconfigure end-to-end encrypted systems to communicate with each other are too numerous for us to attempt complete coverage. We focus on five general areas in particular: user identity, key distribution, user discovery, spam and abuse, and interface design. We try to keep the discussion high-level so that it is largely agnostic to the message exchange architecture (i.e., whether messaging platforms have adopted a standard protocol or opted for client-side bridges).

3.2.1 User Identity

End-to-end encryption is meaningless without sufficient verification of the authenticity of the ends. There are two layers to identifying users:

1. *Cryptographic Identity:* First, how do you know whether a given public identity key belongs to a user Alice's account?

2. *Real-world Identity:* Second, once you have verified the public key attached to Alice's account on this service, how do you know that the user "Alice Appleton" who has contacted you is the Alice Appleton you know in real life?

Both cryptographic identity and real-world identity are needed to assure a user that they are indeed talking to the right person. We discuss each of them in turn.

3.2.1.1 Cryptographic Identity

A user's public key forms their "cryptographic identity". Currently, each messaging provider maintains their own separate public key directory for their userbase. When Alice wants to talk to Bob on a given messaging application (which both Alice and Bob use), Alice's client queries their provider for Bob's public key. The provider looks up Bob in their centralized key database and uses this information to establish an end-to-end encrypted communication channel between Alice and Bob.

Existing key distribution protocols generally require users to trust the provider to store and distribute the correct keys, and are vulnerable to a malicious provider or compromised key server. Interoperability further complicates trust establishment, since users will now have to place some degree of trust in a separate service provider. We revisit this question in greater depth in §3.2.2.

3.2.1.2 Real-world Identity

Tying a cryptographic identity to a real-world identity is an even more challenging problem since security and privacy are somewhat in conflict here. Messaging services vary in the information they ask of users: WhatsApp and Signal both require users to provide a phone number at the time of account registration, though Signal is currently working on username-based discovery so that a user's phone number is known only to Signal [310]. iMessage uses the email address associated with a user's Apple ID by default, but can also be configured to identify a user by their phone number. The Swiss messaging app Threema, on the other hand, identifies users through a randomly generated 8-digit Threema ID, and does not require a phone number or any other information tying a user account to an real-world identity scheme [412]. How is a WhatsApp user to know the Threema user is who they claim to be? At the moment, the only option would be to verify identities through an out-of-band channel (e.g., SMS, email, or meeting in person), which users rarely do in practice [430, 431, 381]. And even if an identity assurance ceremony is performed once in person, many vectors of account takeover have been industrialized by the cybercrime community (such as SIM swapping) while others are widely available to state actors (such as SS7 hacking).

While a real-world identity mechanism like a phone number or email address may give some identity assurance, there are many valid reasons why a user might not want to tie their identity on a messaging service to their real name, so the use of handles or pseudonyms is a desirable property for some messaging platforms to offer. Even in the absence of privacy concerns, users need different handles to cope with congested global namespaces; the 'alice.appleton' at the Cambridge Computer Laboratory becomes 'alicejappleton' on X (formerly Twitter) and 'alice_appleton' on Threads.

Given the importance of identity assurance to interoperable communication, however, it may be that identity assurance and anonymity cannot be reconciled absent out-of-band user verification. Under the DMA, would a service provider be allowed to reject a request for interoperability from a messaging service that does not collect some form of external identity from its users? It is difficult to argue that a service's "level of security" is maintained if platforms are obligated to interoperate with a service that does not rely on some suitably accredited external identity scheme. And if so, would this give smaller services an incentive to remove support for pseudonymous account registration? Such a decision may interact with other EU provisions around identity (such as eIDAS, the European Union's electronic identity verification service) as well as broader policy questions (such as identity escrow and age verification). The eventual outcome might be a creeping 'real names' policy, to the disadvantage of users with a genuine reason to seek anonymity – from political dissidents and stigmatized minorities to survivors of intimate abuse. Such users might have to seek out gray market service providers that have opted

not to interoperate with regulated platforms, which would be marginalized and perhaps pass some stigma to their users.

Importantly, one of the main attractions of these platforms is that it is simple to sign up for them. In most cases, user identity is bootstrapped off of a user's phone number, where a provider sends the user an SMS message with a random string which the user then enters in the app to prove control of said phone number. This ease of use must be preserved in any interoperable scheme, for instance by using an identifier the user already has memorized.

3.2.2 Key Distribution

Contemporary end-to-end encrypted messaging systems maintain platform-specific, centralized key directories to store their users' encryption keys. When a new user registers an account, the user's device generates several public-private key pairs (the precise number and type of key pairs depend on the protocol used) and sends the public keys to the service provider to store in its directory.

From a security standpoint, this reliance on the service provider to store and distribute encryption keys is a major weakness in existing systems. Since the service provider controls the public key directory, a malicious provider could compromise end-to-end security by swapping a user's public key for one under their control, either of their own volition or because they were legally compelled or otherwise pressured to do so. The confidentiality of the communications, then, hinges on trusting that the service provider has provided the correct keys. Existing protocols fail to prioritize this: MLS does not tackle the storage or distribution of keys, only how they might be used to send and receive cross-platform messages.

Interoperable communication further complicates trust issues around key storage and distribution. How would one service provider share the identity keys of its users with another in a way that satisfies user privacy expectations? And how can one platform be certain that the other has shared the correct keys? A platform could conceivably trick a user of another service into talking to a different person than the one with whom they believe they are communicating. Each service provider has no choice but to trust the others. Any open communication ecosystem will need to design explicit and effective controls around accessing, sharing, and replacing keys. Users' identity keys change constantly—every time they delete and reinstall the app or get a new phone, their device generates a new set of keys and the provider updates their directory accordingly. Service providers will need an efficient way of conveying user key changes to other providers. There is some ongoing academic work to develop a decentralized key infrastructure based on existing digital ID systems [407], but these are still in the proof-of-concept stage.

3.2.2.1 Key Transparency

Key transparency is an area of active research that, in theory, would eliminate the need to trust the service provider to share the correct keys. The general idea is that a service provider will still maintain a large key directory, but this directory is now publicly accessible and auditable by independent parties (either a third-party or possibly the service providers themselves auditing each other). A provider cryptographically commits to a user's identity to public key mapping at the time of key generation (such that it cannot be changed without detection), and further periodically commits to the full directory version. Users (more precisely, users' messaging clients) can then verify both their own keys as well as their contacts' keys to detect a provider serving different versions of its key directory to different users. In practice, service providers would likely maintain separate key directories, but other providers would be able to query this directory in a privacy-preserving manner such that an external provider could only query for individual users, along with other privacy mitigations. Note that an auditable keystore system does not *prevent* a key-swapping attack from taking place, it only *detects* when such an attack has happened after the fact.

CONIKS [309], the first end-user key verification design, was proposed in 2016 but suffered from scalability issues due to the frequency with which users would need to check that their key is correct. Several other key transparency systems have been proposed and, in some cases, deployed, in the years since. Subsequent academic research has built on CONIKS, formalizing the notion of a verifiable key directory and mitigating scalability concerns by making the frequency of user key checks depend on the number of times a user's key has changed, along with providing a handful of other practical deployment improvements [112, 287]. For various practical reasons, the largest industry providers have been slow to deploy key transparency. Google released a variation of CONIKS that enables users to audit their own keys in 2017 [371]. More recently, in April 2023 Meta announced plans to roll out large-scale key transparency across WhatsApp [383], though there are still many unresolved implementation questions such as how public audits will be carried out and who the auditors will be.

But deploying a verifiable key directory service at scale across multiple service providers large and small is another matter. The authors of CONIKS neatly outlined several of the main barriers to deployment back in 2016 based on discussions with engineers at Google, Yahoo, Apple, and Signal [289]. The most serious challenges boil down to the difficulty of distinguishing between legitimate and adversarial key changes. When Alice gets a new phone or forgets her iMessage password and resets her keys, how can Alice's service provider convince other providers accessing and auditing the key directory that they have legitimately identified Alice through some real-world identity mechanism, and that these new keys do in fact still belong to Alice? Will Alice need to somehow prove her identity not just to her own service provider, but to all other providers? We are not aware

of any existing key transparency proposals that solve these problems adequately. In the absence of genuine end-to-end identity verification, will WhatsApp simply have to take Telegram's word for it?

Apart from the technical challenges, we cannot assume that platforms will agree on how such a key management service should work. Messaging providers have already made different design decisions around detecting false positives and when to send a user a security warning that one of their contacts' keys has changed, usually in an attempt to avoid inundating users with security warnings. Until very recently, iMessage did not even provide users with an option to manually verify their keys. Their new Contact Key Verification system claims to alert users if an "exceptionally advanced adversary, such as a state-sponsored attacker" has managed to secretly join a chat [65], but few technical specifics of how this works on the backend are publicly available. Many other widely used E2EE services inundate the user with in-chat notifications every time one of their contacts' keys has changed. The complexity of a distributed key verification service, however, makes it even more likely to issue false security warnings due to various synchronization problems.

3.2.3 User Discovery

We can now build on the above discussions of user identity and key management to consider how the process of learning which service(s) a user uses and/or prefers might work.

There are two separate but related design principles, advocated by multiple NGOs, that should be considered in the development of any discovery mechanism [399, 223]:

1. **Separate Communications:** Users must be able to keep their communications on different messaging apps separate if they choose. The Center for Democracy & Technology offered the analogy of using a work email and a personal email, a paradigm adopted by the vast majority of the general public [399]. Prior work has shown that some users likewise use different messaging apps for different purposes [336, 76, 194, 453], a feature that should be preserved in an interoperable world. In other words, users should retain the ability to sign up for a messaging service and opt to receive messages from users on that same service only.

2. **Opt-In by Default:** Related to the first principle, users should be excluded from discovery by default to maintain reasonable user privacy expectations.[3] When a user signs up for a messaging service, they consent to discovery within that service—and

[3] The discussion around opt-in versus opt-out discovery in messaging interoperability is often compared to email since email addresses are openly discoverable and contactable by anyone. But there are well-recognized social conventions and distinctions among email services that do not exist in the messaging ecosystem. For instance, given two of Alice's email addresses, "alice.appleton@gmail.com" and "alice.appleton@company.com", one can reasonably infer which types of communications Alice would like sent to each address without needing to discuss with Alice.

only that service. If they so choose, a user may *opt in* to discovery by all interoperable services or a subset of available services. We see two high-level designs where this could be accomplished: using service-specific identifiers (as with email), or allowing users to change their discoverability preferences in a per-service basis in the settings of the app(s) they use (i.e., Alice can choose to be discoverable on iMessage but not Telegram, etc.).

Keeping these principles in mind, we have two general models for forming user identifiers [234]:

1. **Service-Independent:** Users use the same real-world identifier (e.g., phone number) across multiple services. Alice wants to contact Bob for the first time, but since Bob's identifier is not linked to any one specific service, Alice still needs to figure out where to send her message. Presumably, her messaging app will present some sort of app selection interface for Alice to choose which service to use to contact Bob.[4] If Bob is discoverable on multiple services, either Alice asks Bob which service he prefers out-of-band (e.g., in-person, over email, etc.), or Alice simply selects one of the options from the interface based on her own preferences. If users do not want to select a service manually each time they begin a conversation, then there are various options for automation. The simplest might be for each user to have a priority list (e.g. try Signal, then iMessage, then WhatsApp), just like the negotiation of TLS ciphersuites or EMV credit card chip authentication methods. But if a user wanted family messages on WhatsApp and work messages on Signal, then this would become more complex still.

2. **Service-Specific:** Alternatively, Bob's identifier could be linked in some way to a specific service such that Alice's service provider knows where to deliver messages addressed to Bob. This is analogous to email, where each identifier is scoped to a particular namespace (i.e., alice.appleton@gmail.com and alice.appleton@cam.ac.uk do not necessarily refer to the same person). In the case of messaging, Rescorla has suggested that this could look something like "1.415.555.0123@whatsapp.com" [145].

   Of course, asking the user to enter this mapping would be messy from a usability standpoint, and in many cases the user would still need to go through an out-of-band process to obtain the scoped identifier from the other user. To keep the user experience seamless, this identifier-to-service mapping could be hidden from the user and only used internally. For instance, perhaps each user designates a 'default' or 'preferred' service, which would generally obviate the need to ask the message recipient where to send the message.

[4]Many interface design questions arise here. For now, we focus primarily on privacy concerns in user discovery, and revisit the user experience in §3.2.5.

3.2.3.1 Centralized Directory Service

The tricky part is figuring out how each service provider learns which services are associated with a given phone number. Rescorla and others have floated the idea of a large-scale, centralized database of phone number-to-messenger mappings, similar to how the PSTN (Public Switched Telephone Network) maintains a large database mapping phone numbers to carriers [145]. At a high level, a user record would be added to the directory service or updated at the moment of app installation, once that user has gone through some real-world verification process to their service's satisfaction (e.g., provided a random code sent to their phone number). In theory, this directory might be part of a centralized key distribution service.

But creating such a database for E2EE messaging services is more complicated than the PSTN database for a number of reasons (as Rescorla acknowledges). First and foremost, there is no privacy to speak of in SMS, whether we're talking about discovery or message contents. Any carrier can query the database—which is, of course, one of the reasons SMS is overrun with spam. Second, number-to-carrier is a one-to-one mapping, while number-to-messaging service is a one-to-many mapping (assuming that a significant number of users continue to use more than one messaging service). As discussed above, there are several different design options for selecting a service, including letting the message sender select through an interface, letting the message recipient indicate a preferred service, or perhaps even asking users to provide a ranking of services by context and then choosing the highest-ranked service common to both. Each of these would require a different set of cryptographic protocols to maintain certain privacy-preserving attributes.

In contrast to the PSTN database, in which phone numbers are rarely changed or removed, users may frequently adjust their discoverability preferences for different services, for instance as they make a new acquaintance who wants to use a particular service or receive too many unwanted spam messages from a different service. We will need effective identity revocation mechanisms: if Bob changes his mind and no longer wants to be discoverable by other platforms, how can Bob's service provider communicate this to the other platforms and gain reasonable assurance that they have actually removed Bob and all associated data from their servers?

In particular, the centralized nature of such a service raises several security and privacy concerns, some, but not all, of which can be solved using known cryptographic schemes. The ability of an individual user to send individual query requests to this directory and learn which apps another user is associated with is probably the least concerning since this is the case for several of the largest platforms today. On Signal, for instance, you need only know someone's phone number to learn whether they are also on Signal. While a user can now rapidly retrieve a list of apps used by someone else, instead of having to manually install and test each one, this has minimal impact on the attack surface compared to other

challenges. Presumably the directory would be rate-limited to some extent to prevent mass scraping, though the success of such a mitigation will come down to how rigorously clients are identified.

The most serious concern is that the identity service would know precisely who is talking to whom based on user identity lookups. This might be mitigated through private information retrieval (PIR) or private set intersection (PSI) techniques for anonymous contact discovery, but while academic work has made great strides in improving the scalability of PIR, it is unclear if such schemes are workable on the scale of billions of users. Any contact discovery database will need to update each time a new user joins a platform to maintain basic functional requirements of a contemporary messaging service. When Bob signs up for a new platform, he expects to be able to send and receive messages instantly, not to have to wait 24 hours until the database has been updated to include him. These requirements make PIR impractical given that current academic research relies heavily on server preprocessing of a periodically updated database.

A centralized identity-mapping service would also know all messaging platforms used by every individual, a fact with significant implications for user privacy. Currently, an external user would need to manually query each service in turn to learn whether Alice is a user, limiting any large-scale privacy impact. A related point is the need to make any such lookup service compatible with users' ability to opt in to interoperability on a service-by-service basis. In other words, Alice, who uses S_A, could opt in to discovery by users on S_B, but not S_C. Would S_C still be able to access Alice's records? How could access be controlled such that each provider is only able to query a subset of user records, and who would enforce this? Perhaps instead of a universal lookup service, each set of providers that interoperates shares a lookup server with records of jointly discoverable users, though this still poses many of the same questions around trust, shared hosting responsibility, and scalability.

3.2.3.2 SPIN

To avoid having to use a centralized service, the IETF MIMI working group has introduced a high-level framework for a new identity mapping protocol called SPIN (Simple Protocol for Inviting Numbers) [235] that operates similarly to existing account setups. SPIN assumes that all users are identifiable by their phone number, communicating users already know each other's numbers (as WhatsApp and others operate today), and that users' devices are online at the moment the conversation is started. When Alice wants to send a message to Bob, who uses a different service, Alice's client sends an SMS message to Bob's device requesting the services Bob supports, and Bob's device replies. While this method of identity mapping avoids the problem of using a large-scale directory service, it brings its own downsides. It requires modifications to and the cooperation of the underlying mobile

OS (namely, iOS or Android) in addition to the messaging platforms, requires users to be online to be discoverable, and effectively adds an extra round trip (and possibly a small delay) to the normal communication process [145].

3.2.4 Spam and Abuse

Content moderation is a real challenge in deploying an interoperable network of networks. Detecting and handling spam and abuse effectively is an unsolved problem with plaintext content, let alone encrypted content, let alone across multiple complex distributed systems [306, 380, 37].

3.2.4.1 Existing Techniques

The existing providers have spent years building up their content moderation systems, training complex machine-learning models and hiring human moderators to resolve hard cases and feed back ground truth. The scale is astonishing: WhatsApp bans nearly 100 million accounts annually for violating its terms of service [150]. It is unreasonable to expect that providers will all adopt some new universal content moderation system, unless perhaps mandated to do so by governments.

For now, let us assume an end-to-end encrypted system where only the users can access message contents. In such a setup, providers rely on two primary techniques for content moderation: user reporting and metadata [244, 353].

User Reporting: The most effective content moderation scheme at scale is user reporting [353]. How might reports work for users communicating through two different service providers? Suppose Alice is using S_A, to exchange messages with Bob, who uses S_B. Bob sends Alice an abusive message, prompting Alice to block Bob. Either S_A needs a way of passing this on to S_B, or S_A continues to receive further messages from Bob but simply opts not to display them to Alice. For instance, email handles blocked users by automatically redirecting any future emails from them to a user's spam folder.

Suppose Alice also reports Bob for good measure. Presumably S_B would be responsible for handling the report as Bob is their user, but since Alice has reported him on S_A's interface, S_A will need to pass along relevant information to enable S_B to take appropriate action. When a message is reported, the clients of several E2EE services, including WhatsApp and Facebook Messenger, automatically send the plaintext of the previous five messages in a conversation to the provider along with the reported message to provide additional context [244]. Will this policy be maintained in an interoperable context, such that in order to report Bob, Alice ends up sharing certain personal communications with a different service provider?

All of this also needs to be compatible with existing message franking protocols [196]

(and their metadata-private counterpart, asymmetric message franking [417]). Message franking prevents users from faking abuse reports by cryptographically stamping each message; it has been deployed in Meta messaging services for several years now [154].

Metadata: Since user reporting is inherently retroactive (i.e., the harmful content has already been sent), service providers also rely extensively on metadata to monitor unusual communication patterns in real time. This is most obviously relevant for spam, where a service provider can detect unusual volumes or destinations of messages, but profile or chat descriptions can also be very useful for fighting abuse. WhatsApp bans over 300,000 accounts each month for CSAM sharing based on this approach [447].

From a privacy standpoint, this dependence on metadata raises numerous questions around what data would be viewable by each service provider in an interoperable communication. While the DMA specifies that only personal data that is "strictly necessary" for "effective interoperability" can be shared between providers, this can be quite expansive given providers' reliance on metadata. On the other hand, if a provider is overly limited in what data they can collect, this could have adverse effects on efforts to fight spam and CSAM, negatively impacting the user experience. Len et al. [274] propose to resolve this tension by dividing the responsibility for spam filtering between the sender's provider and the recipient's provider, with the expectation that the sender's provider will do some degree of metadata-based filtering prior to passing the message with metadata removed along to the recipient (or opt not to send the message at all if they deem it to be spam). This would require each platform to rely in large part on an external third-party (the sender provider) for spam filtering, and represents a fundamental reimagining of which parties in a communication channel are responsible for moderating content by shifting much of the filtering to the sender. The feasibility of this proposal in practice will depend on as-yet-undetermined European Commission standards for when and how quickly one provider is able to break off interoperability with another if the level of filtering proves inadequate.

And it cannot be assumed that only two platforms would be involved. If Alice and Bob are on Signal, and Bob adds Carol and Dave on WhatsApp, and Dave then adds Eolina and Firoz on Telegram, and so on, do we maintain message forwarding limits, source tracing, and other moderation techniques end-to-end? WhatsApp, for instance, imposes restrictions on how many times a message can be forwarded to limit viral spread (as this is abused to promote disinformation, such as election-related hoaxes) [445]. Without an inter-provider limit, spam and disinformation can be laundered in just the same way that drug gangs launder their proceeds by sending them along a chain of bitcoin exchanges. But cross-platform forwarding limits will only work if everyone keeps proper records in compatible ways.

3.2.4.2 Content Detection Schemes

While user reporting and metadata analysis are already deployed by most platforms, some may additionally deploy other content moderation schemes which arguably undermine the confidentiality and end-to-end encrypted properties of the messages, whether of their own volition or because they are under a government mandate to do so.

Content Scanning: Suppose a platform using some kind of automated content detection system (such as perceptual hashing [262], which can detect known harmful content in encrypted communications but has been shown to be vulnerable to attack [227, 360]) requests to interoperate with a service that has consciously opted not to use such a system. Will the latter be allowed to refuse this request on the grounds that it would compromise the level of security they provide to users?

And perhaps it is the other way around, where it is the gatekeeper that has deployed some type of client-side scanning mechanism, either voluntarily or under a government mandate. Interoperability can be co-opted by governments to undermine end-to-end encryption in various ways. There is a clear risk, given the Child Sex Abuse ("Chat Control") Regulation before the European Parliament [151] and the UK's 2023 Online Safety Act, that the agencies will mandate client-side scanning on precisely those large platforms on which the interoperability mandate falls. In that case, the mandated client-side bridge becomes a scanning gateway that is in effect under government control. Recent history is littered with examples where law enforcement and the technical community held rather different notions of what violates an existing level of security [16, 18, 216, 429, 17, 275, 29].

Traceability: Message traceability is the idea that platforms should be able to identify the source of a heavily forwarded message [418]. WhatsApp and numerous civil society organizations have spoken out against traceability, arguing that it breaks end-to-end encryption by identifying content users have shared without the consent of the message sender or recipient(s) [446]. Will WhatsApp be required to interoperate with a platform that has deployed some form of message traceability?

Broadly, will the European Commission consider systems using content scanning, traceability, or other moderation techniques to endanger the integrity of a service that does not deploy these techniques, given that they are themselves pursuing such mandates? More to the point, how would a gatekeeper even know whether the requesting service has deployed these or other schemes? Meredith Whittaker, President of the Signal Foundation, has alluded to this challenge, stating that while Signal is open to the general notion of interoperability, they would need to ensure there are no "tricks on the backend" to compromise security or privacy [240].

3.2.5 User Interface Design

Interface design is critical if messaging interoperability is to enhance, rather than degrade, the user experience. We have already seen that interoperable communication inherently presents a reduction in security and privacy compared to communication taking place on a single service. A panelist at a 2023 DMA stakeholder workshop, however, suggested that "we have failed if we have to inform the user of something changing" [150]. This is a well-intentioned but misguided notion that understates the data-sharing implications of interoperability and the depth of the privacy protections promised by platforms. Privacy-preserving cryptographic techniques can only take us so far. Critically, some data *cannot* be hidden in order to allow for sufficiently accurate spam moderation. We will need clear and unambiguous ways of informing the user that their data and messages are leaving their service, and, by extension, that the security and privacy guarantees and features of their platform may no longer apply.

3.2.5.1 App Selection Interface

The Matrix Foundation, Cory Doctorow, and others have drafted preliminary mockups of what the interface for selecting the message recipient's service might look like [302, 125], with Matrix likening it to choosing which application to use to open a file on an OS ("Open with..."). If a user opts to be discoverable on a large number of services (say, $n > 5$), we risk the interface becoming cluttered, but we consider this a comparatively minor problem since a limited number of services will be designated gatekeepers by the EU to begin with. Rescorla [144] suggested the example of email provider selection, where a user is shown a window containing the five or so most common email providers along with a freeform "other" option to enter an alternative provider. Such an interface was mandated by the EU for default browser selection on Windows PCs from 2009–2014, with a random choice order, after an antitrust finding against Microsoft's Internet Explorer.

There are still many open design questions around user discovery that determine what would be displayed to the user. If Bob is discoverable on only one service, should Alice's client default to sending the message to Bob on that service? Or should Alice be shown a pop-up window with just one option that she still needs to manually select in order to ensure she understands and consents to starting a communication channel with Bob's service provider? If Bob is discoverable on multiple services, and one of them happens to be the same service that Alice uses, should the chat default to Alice's service? Suppose Bob has two distinct ongoing communication channels with Alice through different apps Bob uses. Will the messages be intermingled within the same interface (as is the case with iMessage's SMS interoperability), or will they be shown to the user as two separate chats?

If Bob designates a particular platform as his preferred service, would Alice's interface indicate Bob's preferred service (for instance, via a visual nudge to choose it), or would the

chat simply default to it? Would Bob be able to change this after the fact, or would they have to start a new conversation? What if Bob wants his default service to be whatever service Alice is using, and to choose a different one only if Bob is not on Alice's service? Most importantly, can individual platforms make different decisions in response to these questions, or do they need to establish a set of agreed standards for the interoperable user experience? Design questions like these will have an enormous impact on the user experience.

3.2.5.2 Communicating Changes in Security Guarantees

Effectively communicating security and privacy risks to the user is arguably a more difficult problem than designing a new cryptographic protocol for data exchange. We can draw on decades of security usability research showing time and again that users struggle to comprehend and act on data access requests [160, 256, 377] and security warnings [405, 381, 366, 351]. Among other takeaways, researchers have concluded that to be effective, the text of warnings need to communicate clear and specific risks [398]. The user would presumably be given a textual warning with an option to read further details at the moment they opt to be discoverable by an alternative service.

The open question is what distinction, if any, would be shown on a chat-by-chat basis. Matrix has compared this to the WhatsApp Business API [327, 150], which is not end-to-end encrypted when a business uses a third-party vendor (Meta included) to host their API. But while ordinary WhatsApp chats have a small bubble at the top of their chat window informing them that their communications are encrypted end-to-end, a chat passing through the business API simply makes no mention of encryption at all. In other words, WhatsApp "informs" users of the reduction in security through the absence of a typical security indicator, a design which is likely ineffective in building user comprehension. To avoid inadvertent downgrade attacks, it will have to be abundantly clear to a user whether a given conversation is taking place within their own service only, or across multiple third-party services [19]. And visual indicators are no silver bullet: Signal recently removed SMS support for Android, citing, among other reasons, the need to avoid inadvertent user confusion given that SMS and Signal messages were both sent in the same interface. As Signal put it, they "can only do so much on the design side to prevent such misunderstandings" [331].

We know from existing usability research that whether a user takes a warning seriously depends heavily on the design [425]. In one example back in 2013, Akhawe et al. [25] found that Mozilla Firefox's SSL warnings were significantly more effective than those in other browsers due to variations in design: around 70% of users clicked through Chrome's SSL warnings, while only one-third clicked through Firefox's warnings. Chrome has since invested heavily in redesigning their SSL warnings based on what Felt et al. [161] termed

"opinionated design", meaning that the user should be nudged towards the option perceived by the designer to be superior through visual cues (instead of text explanations). In practice, this is typically accomplished by varying colors to encourage the user to choose the safer option or requiring the user to click through an extra window before advancing. Of course, in industry such ideas have been co-opted to create interfaces that nudge users to select an option that is in the company's interest—the more well-known term for these design strategies is now "dark patterns" [193].

3.2.5.3 Blue Bubbles and Green Bubbles

Fortunately, we already have a real-world case study demonstrating the impact of interface design choices on user decisions in an interoperable setting. While Apple's iMessage interoperates with SMS/MMS, Apple uses visual contrasts to make the difference abundantly clear to the user. When two iPhone users communicate through Apple's default Messages app, the sender's messages appear blue (indicating that the communication is taking place over Apple's iMessage). In contrast, while an iPhone user is able to use the same interface to talk to an Android user, the iPhone user's sent messages now appear green, indicating that the messages are being sent over SMS. This distinction has given rise to social pressure to use an iPhone over an Android phone, thereby further consolidating Apple's market control, particularly over younger generations [349, 413]. In the words of one user, "If that bubble pops up green, I'm not replying" [349].

While the bubble color is not the only difference the user experiences (green bubbles lack certain iMessage features like emoji reactions, group naming, typing indicators, etc.), the rapid propagation of the blue versus green bubbles phenomenon throughout popular culture demonstrates just how effective these types of visual cues can be in branding one option as "good" and the other "bad"—indeed, from Android's perspective, perhaps a little too effective [375].

The ongoing bubble war dispels any notion that interoperability on its own will defeat network effects. Far from opening up Apple's "walled garden", iMessage/SMS interoperability appears to have further solidified Apple's market power. At the same time, however, Apple has an obligation to its users to inform them of the difference for fundamental security reasons. The same is true even when two E2EE services interoperate: as Meredith Whittaker recently pointed out, simply being end-to-end encrypted is not "an end in itself" [311]. Messaging services have widely varying privacy policies, cloud backup schemes, jurisdictions in which they operate, receptiveness to client-side scanning, levels of cooperation with foreign governments, and so on. Rather, the goal must be to preserve a broader understanding of privacy and situational awareness of possible attacks. The wicked question here, and one for which we have no answer, is how to convey accurately and clearly to the user what is happening when they opt to interoperate with another

service, while not needlessly discouraging them from doing so.

3.3 Current Status

Here, we discuss the current status of messaging interoperability in the EU under the Digital Markets Act. We note that as this chapter and section are concerned with an ongoing regulatory mandate, the information here may be out of date and is only accurate as of the publication date of this thesis.

WhatsApp and Meta, the DMA's two gatekeepers, formally published technical reference offers for other messaging services interested in interoperating in March 2024 and September 2024, respectively [313]. Matrix (along with its flagship client Element) has expressed the greatest public interest and formally requested interoperability with WhatsApp on March 6, 2024 [140]. To the best of our knowledge, no other services have gone as far as Matrix in formally beginning the process of interoperating. Signal and Threema have publicly stated that they have no plans to pursue interoperability with Meta services due to concerns over user data sharing [288], with Signal President Meredith Whittaker stating in February 2024 that for Signal to interoperate even with "a Matrix service would mean a deterioration of our data protection standards" [288].

Moreover, Matrix and other smaller services which were initially interested in pursuing interoperability with WhatsApp and Facebook Messenger have run into financial limitations when weighing the benefits of interoperability for their users with the resources required to properly implement it. Since Meta is only legally obligated to roll out interoperability to users physically located within the European Economic Area (EEA), they have opted not to release interoperable features beyond the EEA. This appears to have come as a surprise to Matrix, who wrote in a blog post that the restriction to EEA users means that it is no longer "financially viable" for Matrix clients or most other smaller platforms to dedicate the resources to deploying interoperability with Meta services [210]. As of September 2024, Matrix was seeking collaborations with more well-resourced organizations to sponsor production-grade interoperability [210].

There have additionally been minor technical disputes over Meta's proposed design. In particular, discoverability (also termed *reachability*) was a major open question left unresolved by the language of the DMA. While WhatsApp originally designed their scheme such that WhatsApp users would be unreachable by third-party service by default [315], following discussions and pushback from partner services Meta agreed to change their policy and allow Meta users to receive notifications from other services without explicitly opting in [313, 210]. Other points of contention have largely focused on metadata sharing—specifically, the perception by smaller services that too much metadata is being sent to Meta. To detect and prevent spam and scams, WhatsApp requires that any

interoperating servers provide client IP addresses when establishing a connection, which Matrix among others have deemed “problematic” [140]. Metadata sharing is a significant open issue on which Meta and other services remain at an impasse. Incidentally, the client-side bridging architecture and overall design have proved to be the least disputed aspects of interoperating two different messaging services, with all Matrix traffic converted into WhatsApp’s API [210].

Finally, from a usability perspective there are already indications that the user experience will be far from seamless. Currently, WhatsApp interoperability only works with one device at a single time (since only one cryptographic identity can be associated with a user account in the basic Signal protocol), and thus a user cannot interoperate with WhatsApp on both their mobile client and desktop [313].

3.4 Summary

Having explored in depth what an interoperable messaging solution might look like in this chapter, we can see that the core message communication protocol is just the beginning. Interoperable systems need protocols to support the many other features that make secure communication possible, including cross-platform user authentication and tracking data exchanges between platforms. A harbinger of the technical difficulty may be the fact that Meta has been trying to interoperate WhatsApp and Facebook Messenger since 2019 – and this is for two services owned and operated by the same company, which has a strong business incentive to make it work [291, 79]. Meta’s President of Global Affairs, Nick Clegg, acknowledged in a 2021 interview that inter-platform interoperability is “taking us a lot longer than we initially thought” [79].

In any serious discussion of security and privacy trade-offs, the yardstick needs to be how much the attack surface has increased compared to existing systems. For instance, disappearing messages, a feature allowing a user to set messages to be automatically deleted after some specified period of time, have been held up as an example of a feature that is difficult to ensure in an interoperable service [109, 215]. Even if both services ostensibly offer the feature, each has no guarantee that the other has actually complied with the deletion request. When the subject came up at the European Commission’s stakeholder workshop, the panelists’ response was that a user never really knows what happens to their communications once they’ve left their device anyway—the message recipient may take a screenshot, have malware on their device, and so on [150]. This rejoinder dodges the reality that interoperability would add a substantial new attack vector in the form of the interoperating service. That disappearing messages, and indeed many of the challenges discussed throughout the paper, are already concerns in existing systems is no reason to make things worse.

Finally, the discussions in this chapter should not be interpreted as an argument against *any* form of messaging interoperability. E2EE messaging interoperability is already mandated in a major geopolitical region with implementation work well underway, and the question is no longer *whether* to implement interoperability but *how* to do so. As such, this chapter interweaves discussion of two distinct objectives: an idealized, privacy-preserving interoperable design that will likely take several years to design and implement, and a shorter-term solution satisfying the requirements of the EU's Digital Markets Act. The discrepancies between these two objectives are the prevailing theme of this chapter, and serve as a prime case study of the ways in which system design choices can either bolster or undermine E2EE communications security. In subsequent chapters, we will build on this case study by analyzing very specific design choices around key storage and authentication in E2EE services.

Chapter 4

KeyDroid: A Large-Scale Analysis of Secure Key Storage in Android Apps

Having surveyed broad considerations for building an E2EE messaging service in Chapter 3, we now turn to a handful of specific design decisions that can have a critical impact on the security of an E2EE system. In this chapter, we discuss possibilities for cryptographic key storage with particular emphasis on key storage in mobile devices.

Mobile devices store highly sensitive user data ranging from private health information and payment credentials to personal photographs and correspondence. At the same time, mobile handsets are regularly lost or stolen, making data stored on devices vulnerable to an adversary with physical device access. Similarly, sensitive data could be accessed by an adversary who is able to compromise the main operating system (OS), such as malicious third-party apps which circumvented app store vetting processes or were independently downloaded by users [464, 439].

Modern encryption methods such as TEEs or SEs may provide data confidentiality and integrity against such threats, but data is only as secure as the cryptographic keys used. Keys stored in a software keystore (e.g., Java's Bouncy Castle keystore) are vulnerable to memory-extraction attacks [406, 285] where an adversary with full control over the operating system or physical access to the device can retrieve the decryption keys or other sensitive data through a memory dump of device RAM.

Almost all Android handsets now offer some form of hardware-backed storage, and recent premium models of Android smartphones such as the Google Pixel devices contain additional hardware in the form of a separate secure processor (commonly known as a secure element, or SE) [213, 58, 257]. In Android, the SE is called the `StrongBox Keymaster` [50]. Android has offered the Android Keystore system [58] as its public trusted hardware API for developers since 2011. The Android Keystore API uses the device's TEE by default but also offers developers the option of requesting StrongBox instead.

Unfortunately, there is currently a lack of empirical evidence on when and how developers use secure hardware in practice. Secure hardware is only useful if it is actually used, and the Android Security team acknowledges that apps need to explicitly use these APIs in order to see a security benefit as Android's historical Java cryptography APIs use a software-backed keystore by default [307]. Furthermore, while hardware-backed keystores provide significant security benefits, runtime performance is a critical consideration for mobile developers. More advanced forms of secure hardware (e.g., StrongBox), tend to come with an accompanying performance hit, as acknowledged at a high level in Android's documentation [50], but to date there has been no publicly available empirical data on hardware keystore performance to the best of our knowledge. At the same time, however, Android is pursuing public initiatives to encourage wider adoption of secure hardware such as the Android Ready SE Alliance (see §2.2.3). The lack of empirical evaluation of performance and existing usage patterns is a major barrier to encouraging more widespread adoption: without detailed performance statistics, developers cannot make informed choices about the trade-offs between security and performance for their use case.

In this chapter, we conduct the first comprehensive and systematic study of secure credential storage in Android, analyzing both the contemporary usage and performance of key storage schemes. We compile and analyze a dataset of 490,119 Android applications between October 2023 and August 2024, extracting data from 64 API calls relevant to key storage in Android. We find that 56.3% of apps report collecting sensitive data as part of the Play Store's data safety labels do not use any form of trusted hardware, and only 5.03% contain a reference to the SE API. Moreover, these usage figures represent an upper bound on security within the Android app ecosystem as it is not possible to detect at scale whether apps which contain at least one reference to the Android Keystore API are using it to secure all sensitive and relevant credentials. In particular, of those apps that do use the Android Keystore API, 94.7% of key initializations are located in third-party components, indicating that use of the Keystore API may be due to using a general-purpose library rather than a conscious choice to use hardware-backed key storage. Furthermore, we find that 8.5% of keys generated in the Android Keystore explicitly disable Android's randomized encryption requirement (i.e. IND-CPA, the requirement that a ciphertext must be indistinguishable under a chosen-plaintext attack), indicating that secure defaults are not enough to enforce security guarantees.

Having measured the usage of secure key storage across the Android app ecosystem, we investigate the runtime performance of common cryptographic operations using the hardware-backed key storage APIs to consider whether performance overhead may discourage adoption. We find that the performance of TEE-backed key storage is viable for the vast majority of common app use cases and is noticeably different from a software-backed

keystore only for large payloads greater than 5 MiB. StrongBox introduces a far more significant performance hit. For instance, encrypting a 1MiB message with AES-GCM takes around 3 seconds and simply *generating* asymmetric keys in StrongBox takes over 9 seconds in Google's flagship Pixel 8 device, a runtime which may be prohibitive even for security-conscious apps. Even so, StrongBox's performance has improved significantly since first introduced by Android in 2018 and is viable for use cases involving small payloads, such as using StrongBox to encrypt a key generated by a keystore with less overhead. To the best of our knowledge this is the first time comprehensive performance measurements of trusted hardware in mobile devices have been published, providing Android developers with empirical evidence to make informed decisions for their particular use case.

This chapter is based on our paper "KeyDroid: A Large-Scale Analysis of Secure Key Storage in Android Apps." For this chapter I adapted the text and figures from the original paper to suit the dissertation. I performed all data collection and conducted the experiments and subsequent analysis, and was the primary author of the text. Ross and Alastair contributed towards the development of the ideas and their presentation.

4.1 Key Storage in Android

There are two main forms of hardware-backed key storage in the Android ecosystem, each offering different security properties: (1) a trusted execution environment (TEE), available in Android through the `Android Keystore` API [58] and (2) a secure element (SE), termed `StrongBox Keymaster` [50] and provided as a subset of the Android Keystore API. We discuss each of these in turn below as different forms of secure hardware vary in degree of isolation from the Android OS, and hence the attacks they protect against. Throughout the rest of the paper, we use the terms "Android Keystore" and "Keystore" to refer specifically to Android's trusted hardware API (either TEE or SE).

4.1.1 Software-Backed Key Storage

Java's Cipher API [54] and the Java Keystore API [55] using a software-backed provider (either Bouncy Castle or AndroidOpenSSL, also known as Conscrypt [63]) are both examples of software-backed keystores within Android that have been available since Android's inception in 2007. On all Android devices today, if no keystore provider is specified when using Java's cryptographic APIs (namely `java.security.*` and `javax.crypto.*`) Android defaults to using a software-backed keystore even if the device supports hardware-backed key storage [428, 59].

Prior work investigating Java cryptography APIs has also observed that these libraries have an unfortunate tendency to use the weakest ciphers as defaults (ECB mode with symmetric encryption being the most pervasive example) [174, 139, 95]. Such choices shift

the responsibility for achieving an adequate security level from the API provider to the developer.

4.1.2 Hardware-Backed Key Storage

4.1.2.1 Trusted Execution Environment

In Android, a device TEE's OS is named Trusty [51] and communicates with the Android OS through requests forwarded through the Android Keystore interface to the TEE, referencing keys by a string alias. In Android, the TEE is located on the main processor, which is divided into the Android OS and the Trusty OS [51] (i.e., the normal world and the secure world).

In Android, a TEE has been available since Android 4.3 (API level 18) was released in 2013, with new features added over the years since [58, 60]. The initial version of the Android Keystore only supported asymmetric cryptographic operations and did not add support for symmetric keys until Android 6.0 (API level 23) in 2015 (approximately two years after the initial release date). Hardware-level key attestation, the ability to verify that keys are indeed stored in a hardware-backed keystore, and other more advanced features were introduced in Android 7.0 (API level 24) [41].

In addition to hardware-derived security benefits, the Keystore API makes deliberate design choices that provide an increased level of security in practice when compared with older Java APIs. The API explicitly disallows certain insecure key configurations, such as symmetric encryption with a constant initialization vector, and offers more secure defaults.

4.1.2.2 Secure Element

Most premium Android smartphones include a secure element (SE). Android's public SE API is termed the `StrongBox Keymaster` [50] (henceforth abbreviated as StrongBox) and has been available to external developers since Android 9.0 (API level 28) was released in August 2018 [58]. An SE was first introduced in Pixel devices, Google's flagship device line, with the Titan M chip (Google's in-house secure element processor) in the Pixel 3 in 2018 [461]. This was upgraded to the Titan M2 chip beginning with the Pixel 6 in 2021 [257]. System-on-chip (SoC) hardware, or integrated secure elements (iSE), qualify as providing StrongBox support as long as they meet the requirements above [58]. In 2021, Google's Pixel 6 introduced Google Tensor, a system-on-chip (SoC) that is isolated from the main processor but also has its own CPU and ROM, among other features [257]; Google Tensor interfaces with the Titan M2 chip.

Google's documentation describes StrongBox's performance as "a little slower and resource-constrained (meaning that it supports fewer concurrent operations) compared to TEE", and recommends StrongBox for developers who "want to prioritize higher security

guarantees over app resource efficiency" [58]. Due to these performance drawbacks, the Android Keystore API is structured so that developers must explicitly opt in to using StrongBox even when the application is running on a device that contains a SE.

4.1.3 Related Work

Android App Analysis: Most prior work studying security and privacy in Android apps has used metrics such as permissions requested [159, 277, 365] and traffic analysis [156, 358, 342] and has often overlooked data storage, even when investigating overall app security [258]. For instance, Gilsenan et al. [181] studied security issues in two-factor authentication (2FA) apps and recommended that apps use the Android Keystore, but did not investigate how apps actually do store their keys.

Egele et al. [139] studied cryptographic misuses in Android applications in 2014, but looked only at the software-backed Java Cryptographic Architecture APIs (presumably due to the timing of the work, since the initial Android trusted hardware API was only released in 2013). They noted at the time that both Java and Android JCA APIs allowed a developer to specify only the encryption algorithm (e.g., AES), in which case Java and Android used ECB mode with PKCS7Padding as the default. Focardi et al. [174] similarly analyzed the confidentiality and integrity properties provided by various software-backed Java keystores in 2018.

Hardware API Usage: There have been a handful of studies focusing on particular subsets of hardware-related API usage in Android. Bianchi et al. [87] conducted an empirical survey of Android's Fingerprint API and found very low adoption rates, with just 424 of 30,459 popular apps scanned using the API. Imran et al. [218] ran a keyword search for the key attestation API on a subset of apps in sensitive categories (e.g., finance, communication, medical), finding that of 112,886 apps only five use key attestation. Concurrently to this work, Bove [103] conducted a high-level study on various TEE-based Android APIs (including the Biometrics and Digital Rights Management APIs) on a randomly sampled subset of Play Store apps and found that 32.0% of apps analyzed contained a call to the Keystore API (excluding gaming apps), but only measured the binary question of whether an app contained any Keystore API call without investigating usage specifics. Additionally, a particular focus of this chapter is comparing TEE and SE APIs in both usage and performance.

Coojimans et al. [124] systematized high-level security properties of Android key storage options in 2014, observing that while Android's TEE-backed key storage provides device binding (i.e. prevents keys from being extracted from the device) where software keystores are vulnerable, the implementation of the TEE keystore made it possible for an attacker with root permissions to use other apps' keys (i.e. did not effectively provide app-binding).

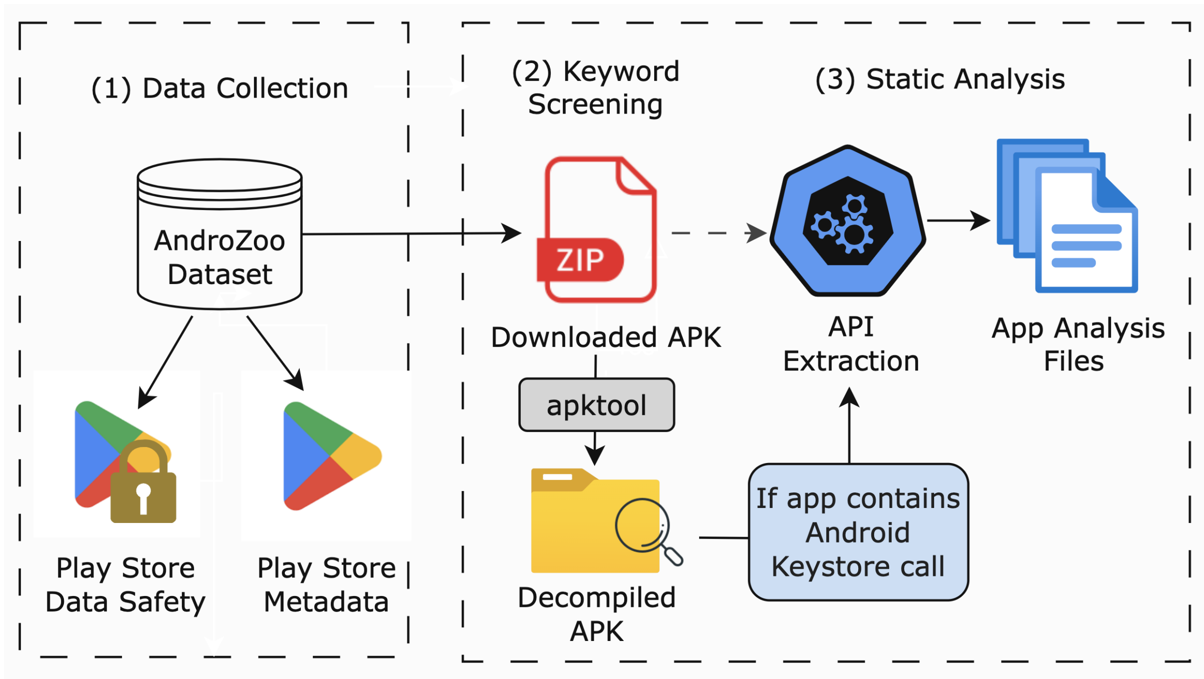

Figure 4.1: **KeyDroid Stages:** We (1) scrape Play Store metadata and data safety information for all apps in the AndroZoo dataset with at least 10,000 downloads and (2) decompile each app and pre-screen for any relevant API references. If an app contains a reference to the Android Keystore API, we run KeyDroid, our in-depth static analysis tool, to generate the app call graph and extract all API references, arguments, and call packages.

This chapter expands on this discussion to consider new forms of hardware (namely, SEs) that were not available when Coojimans et al. surveyed Android key storage in 2014.

Trusted Hardware Performance: To the best of our knowledge, Android does not provide official quantitative assessments of trusted hardware performance. There has been a small amount of prior work measuring specific aspects of trusted hardware performance in Android as supporting experiments demonstrating the viability of a proposed cryptographic scheme. Hugenroth et al. [213] measured the performance of HMAC execution in SEs on Android and iPhone devices to confirm their proposed key stretching scheme was feasible on contemporary devices, observing that time elapsed increases linearly with input length and that a 10 KiB payload takes approximately 1 second to execute in the Pixel 3 SE. This chapter presents the first comprehensive, longitudinal analysis of the performance of various key storage schemes, measuring the comparative performance of all widely used ciphers across the three major key storage options for developers (a software-backed Java Keystore, Android's TEE Keystore, and Android's StrongBox SE API).

4.2 Methodology

We begin by describing our process for collecting our dataset of Android apps and analyzing these apps with respect to API usage. Figure 4.1 provides a high-level overview of all app analysis stages. We further describe our methodology for testing the runtime performance of different keystores across common cryptographic operations.

4.2.1 Dataset Selection

We use the publicly available AndroZoo dataset [31] as our source for Android applications. AndroZoo provides the raw APK files, while all other app metadata and data safety information was scraped directly from the Google Play Store. We initially identify 8,804,118 apps in the AndroZoo dataset from the Play Store marketplace which were crawled on or after July 2013, when Android's trusted hardware API was first released to developers, though this number includes different versions of the same app and apps no longer available for download. We necessarily only consider free apps since the AndroZoo dataset does not include paid apps.

For each app provided in the AndroZoo dataset that passed preliminary filtering, we scrape the Play Store between October 2023 to March 2024 to filter for apps currently available at the point of scraping with at least 10,000 downloads. We collect other relevant app metadata at the same time, resulting in a dataset of 490,119. We were able to successfully download and decompile almost all of these apps, leaving us with a revised dataset of 486,234. We record the following metadata for each app: app package ID, title, number of installs, developer name and email, Play Store genre, release date of the latest version (release date of initial version is not available), and version number.

We download the Android Package (.apk) archive file containing the app source code, metadata, and other resource files for each of these 486,234 apps. When there are multiple versions of the same app (as identified using Android's APK package name) available in the AndroZoo dataset, we use the most recently crawled version.

4.2.2 Play Store Data Safety Labels

App key storage is only a concern if the app processes sensitive or confidential data. Since July 2022 Google has required each app listed in the Play Store to complete a data safety form containing self-reported information from the app developers on what types of user data the app collects and shares with third-parties, and for what purpose; this includes data collected by third-party libraries. For instance, the Signal messaging app notes that it collects only a user's phone number for "app functionality and account management", and does not share data with third parties [397].

In this thesis, we use the Play Store's data safety label information to determine which apps process sensitive data, and therefore which apps may be expected to make use of secure key storage.

4.2.2.1 Published Data Safety Information

According to Google's developer documentation, "all developers that have an app published on Google Play must complete the data safety form" (including apps that self-report not collecting user data) [396]. In practice, we find that only 74.47% (342,872/460,362) of apps have submitted a data safety form at the time of scraping in March through April 2024[1]. Khandelwal et al. [253] had previously conducted a large-scale analysis of Play Store data safety labels in May 2023 (approximately one year earlier) and found that only 46.8% of apps reported any data, so we note the percentage of apps providing a data safety label has increased significantly from approximately a year earlier, though it is still noticeably far from satisfying the Play Store mandate.

For apps that have data safety information, we classify each app as "sensitive" or "benign" based on the types of data the developer has reported. Google uses 14 high-level data type categories, such as location, financial information, audio files, etc. [396] We consider an app to be sensitive if it collects any information from at least one of 12 of these 14 data types. We exclude the final two categories, "App info and performance" (defined by Google as crash logs and other app performance data) and "Device or other IDs" (e.g. MAC address or Firebase ID), since we are interested in whether developers are intentionally collecting sensitive user data relating to specific individuals, which we broadly define as user-provided data. Based on this classification, we find that of the 342,872 apps reporting data safety information, 46.75% are sensitive (and therefore 53.25% are benign).

4.2.2.2 Developer Self-Reporting

Google uses a somewhat counter-intuitive notion of what constitutes data collection: instructions to developers state data is considered to be collected if it is transmitted "from your app off a user's device" [396], and user data that is only processed and stored locally does not need to be reported as "collected". In short, it is possible that an app that processes sensitive user data locally (and may therefore be expected to use some form of hardware-backed key storage) yet this app would not be listed as collecting sensitive data.

Therefore, by using the information in the data safety labels there is a risk our analysis excludes apps which do in fact process sensitive data. Nevertheless, we argue that there is

[1] The slight difference in number of apps for which we attempted to retrieve a data safety label (460,263) vs. number of apps downloaded and decompiled (486,234) is due to apps that were available in the Play Store at the time we began scraping apps themselves in October 2023 but had been removed by the time we began scraping app data safety pages in March 2024.

much to be gained from understanding how the Keystore API is used by those apps which state they process sensitive data: if these apps do not make use of hardware-based secure storage, we hypothesize that it is unlikely that those apps which do process sensitive data, but do not declare it in their data safety label, process such data securely.

There are both benign and malicious reasons for developers inaccurately reporting their use of sensitive data. For example, developers may be unaware of the data collected by third-party libraries or wish to avoid highlighting the data their app collects in their submission, and thus may understate data collected. Conversely, it is also possible that developers may err on the side of *overstating* the sensitivity of the data they collect to ensure they are in compliance with Play Store policies. In principle, Google can often verify whether an app collects sensitive data (or not) and spot any differences between app behavior and reported collection. If discrepancies are found, Google has the ability to block app updates or remove the app from the Play Store altogether. We are unable to determine the extent to which such verification and enforcement takes place and therefore validate the correctness (or otherwise) of the data safety label information.

4.2.3 Static Analysis

To reduce computational load, we use multiple layered static analysis techniques to filter for references to Android's trusted hardware APIs and extract relevant API calls. We begin by executing a basic keyword search across all APKs in our dataset, and then perform more in-depth static analysis on any APKs flagged as relevant.

Keyword Filtering. Since analyzing the call graph is very resource-intensive, to filter candidate apps we first decompile each .apk file using apktool [64] and run an initial grep search for any call to the Android KeyStore API (`android.security.keystore`). After filtering out any apps that do not contain at least one reference to the Keystore API, we are left with a dataset of 303,948 apps.

4.2.3.1 Inter-Procedural Call Graph Analysis

To analyze the bytecode of the 303,948 apps flagged as having at least one relevant API call, we use Soot [2], a well-known framework for inter-procedural static analysis [276] also used by similar related work. We experimented with using FlowDroid and other static analysis tools that more accurately model the Android lifecycle (e.g., by detecting implicit callback methods such as `onCreate` or `onClickListener`) but found that the runtime was sufficiently large as to make it infeasible for a dataset of our size, in large part due to its iterative callback calculation, which recomputes the call graph each time a new callback is encountered. Prior work [457] showed that Flowdroid did not finish app call graph generation on 24% of apps even with a timeout of 5 hours, consistent with our own

observations, and so we ultimately determined Soot offered the right balance of accuracy and efficiency.

We allocate each APK 10GB RAM and set an automatic timeout of 30 minutes. Our analysis tool begins by generating the call graph of the APK to determine the context for a particular API reference. To keep runtime manageable, we assume that all methods are reachable while generating the initial call graph, and conduct a custom reachability analysis (described in more detail below) tracing backwards from a method of interest along the method call chain.

We search for 64 distinct API calls, including all methods from the primary KeyStore API as well as other Android cryptography APIs that in turn call the KeyStore API, such as `androidx.security.crypto.MasterKey` [44] and `android.security.keystore.KeyProtection` [43], and the primary methods from the Java KeyStore [55] and Cipher APIs [54]. The Java cryptographic APIs allow developers to specify a keystore provider, and so we check these to see if developers are referencing the Keystore API indirectly. The full list of specific API methods searched for is available in our dataset in Table 4.1. For each API call identified, we collect the full method signature of the calling method, including associated package and class names, record the object on which the method is called (i.e., register value), and extract all parameter values by applying backwards program slicing [442]. As part of our reachability analysis, we conduct a backwards breadth-first search and trace each method containing a relevant API call backwards through the call graph for up to 1,000 nodes, recording all possible paths.

4.2.3.2 Package Analysis

We are particularly interested in determining whether a particular API call is located within the main application code or whether it is part of a third-party library. First-party usage indicates that developers have consciously chosen to store cryptographic key material in trusted hardware, while for certain third-party libraries developers may be unaware that this is even occurring.

To determine call context, we classify packages as first- or third-party by checking whether the same package is called by other APKs, following similar methodology used by Oltrogge et al. [342] (described in more detail below). While there are a small number of public datasets of third-party library signatures, we find that these are generally too outdated or otherwise incomplete to be fit for purpose (e.g., LibRadar [284] was last updated in 2018).

We collect all packages containing a call to the Android Keystore key generation constructor `android.security.keystore.KeyGenParameterSpec.Builder(String keystoreAlias, int purposes)`. If a package is referenced by multiple APKs from different developers, we consider it to be third-party; otherwise, if it is referenced by only a single

APK or by multiple APKs from the same developer, we classify it as a first-party package.

4.2.3.3 Obfuscation

We observe a significant amount of obfuscation of package names where package names are shortened and anonymized (e.g., `o8` or `q1.x.a`), likely due to built-in obfuscation techniques available to developers in Android Studio and other widely used development tools.

Different packages may share the same obfuscated name, and so we exclude obfuscated packages from party analysis. To identify non-obfuscated package names, if a package name has at least one sub-component (i.e., character string separated by periods) of at least three characters in length, we consider it to be an authentic (non-obfuscated) package name.

4.2.3.4 Reachability

Our call-graph generation methodology described above errs on the side of favoring false positives over false negatives (i.e., we would prefer to include a relevant API call that may be unreachable than to exclude a call that is used). To reduce the risk of false positives, once we have classified all packages as first-party or third-party, to determine whether a particular API call is reachable we trace backwards through the recorded call paths along the control flow. If there exists at least one path containing a call to first-party source code, we consider the API call to be reachable.

4.2.4 Performance Measurements

From the average developer's perspective, perhaps the most important consideration when choosing among different key storage APIs is performance. A natural corollary to surveying the usage of secure key storage is to investigate key storage runtime performance, particularly among different forms of hardware-backed key storage.

To conduct systematic performance measurements we wrote a benchmarking test application that performs symmetric and asymmetric key generation, message encryption, and message signing following canonical examples provided in Android documentation and Android's developer blog [42, 232]. We use AWS Device Farm [6] to run our test application across a variety of Android devices.

4.2.5 Limitations

Here we acknowledge the following limitations of our analysis and describe steps taken to mitigate these limitations.

Accuracy of static analysis: As with prior work in Android app analysis, our research is subject to the inherent technical limitations of static analysis. Given that we only have access to packaged bytecode instead of the original source code, we cannot guarantee that certain source code components have not been obfuscated, though it is unlikely that this would be the case for Android system APIs. Static analysis cannot reliably detect whether a particular component is executed at runtime (i.e., dead or legacy code), but this is a natural trade-off with the scale of our work. Modern compilers and widely used app optimization tools are highly effective at removing unused source code and so we anticipate app bytecode is unlikely to contain unreachable code at the point of our analysis. Dynamic analysis would further preclude studying certain categories of apps, such as financial apps, since we cannot create test financial accounts for regulatory reasons.

Necessity of high-level analysis: The scale of our work (downloading and analyzing around half a million apps) necessarily means that our analysis will be comparatively high-level. In particular, static analysis is unable to automatically detect the semantic application context in which a trusted hardware API call occurs, including what particular data is being stored within the hardware element and how keys generated are being used, or to guarantee that the flagged API call is used to protect sensitive data at runtime (e.g., an app might import a marketing analytics API that in turn references the Keystore API for processing analytics data). However, in our work we are primarily interested in which apps choose *not* to use trusted hardware, particularly SEs, and why. Our results provide an empirical upper bound on secure key storage usage and provide comprehensive data on API usage and performance across the Android ecosystem as a whole.

4.3 Secure Hardware Usage in Android

As the first step in our work, we conduct a comprehensive survey of all Android API calls relevant to key storage or trusted hardware, collecting arguments provided and relevant context (e.g., class and package name in which the call occurred). While we make every effort to retrieve the parameter argument via constant propagation in cases where static analysis initially returns the register value, this is not always possible and thus in the results below the parameter total for a particular API method call is generally lower than the method call total shown in Table 4.1.

We further note that unless otherwise specified, statistics for API calls discussed throughout this section are not necessarily distinct: if a particular API call is located within a third-party library, this call configuration (e.g., parameters) is then duplicated in our findings for each call to this library (including across separate apps). We intentionally consider duplicates in our findings since our purpose is to understand the state of Android security and keystore usage in the wild, though for certain highly relevant calls we will

Keystore API Method	**Count**
void ¡init¿(java.lang.String,int)	278,567
android.security.keystore.KeyGenParameterSpec$Builder setEncryptionPaddings(java.lang.String[])	235,719
android.security.keystore.KeyGenParameterSpec$Builder setBlockModes(java.lang.String[])	224,169
android.security.keystore.KeyGenParameterSpec$Builder setKeySize(int)	166,379
android.security.keystore.KeyGenParameterSpec$Builder setUserAuthenticationRequired(boolean)	48,150
android.security.keystore.KeyGenParameterSpec$Builder setDigests(java.lang.String[])	48,095
android.security.keystore.KeyGenParameterSpec$Builder setCertificateNotAfter(java.util.Date)	44,087
android.security.keystore.KeyGenParameterSpec$Builder setCertificateNotBefore(java.util.Date)	44,062
android.security.keystore.KeyGenParameterSpec$Builder setRandomizedEncryptionRequired(boolean)	30,245
android.security.keystore.KeyGenParameterSpec$Builder setIsStrongBoxBacked(boolean)	24,656
android.security.keystore.KeyGenParameterSpec$Builder setUserAuthenticationValidityDurationSeconds(int)	23,946
android.security.keystore.KeyGenParameterSpec$Builder setKeyValidityForOriginationEnd(java.util.Date)	15,334
android.security.keystore.KeyGenParameterSpec$Builder setSignaturePaddings(java.lang.String[])	9,313
android.security.keystore.KeyGenParameterSpec$Builder setUserAuthenticationParameters(int,int)	8,974
android.security.keystore.KeyGenParameterSpec$Builder setInvalidatedByBiometricEnrollment(boolean)	6,629
android.security.keystore.KeyGenParameterSpec$Builder setAlgorithmParameterSpec (java.security.spec.AlgorithmParameterSpec)	5,531
android.security.keystore.KeyGenParameterSpec$Builder setAttestationChallenge(byte[])	2,724
android.security.keystore.KeyGenParameterSpec$Builder setKeyValidityEnd(java.util.Date)	1,295
android.security.keystore.KeyGenParameterSpec$Builder setKeyValidityStart(java.util.Date)	1,088
android.security.keystore.KeyGenParameterSpec$Builder setUnlockedDeviceRequired(boolean)	383
android.security.keystore.KeyGenParameterSpec$Builder setKeyValidityForConsumptionEnd(java.util.Date)	230
android.security.keystore.KeyGenParameterSpec$Builder setUserAuthenticationValidWhileOnBody(boolean)	93
android.security.keystore.KeyGenParameterSpec$Builder setUserConfirmationRequired(boolean)	47
android.security.keystore.KeyGenParameterSpec$Builder setUserPresenceRequired(boolean)	38

Table 4.1: Usage count of Android Keystore API methods across all apps in the Play Store.

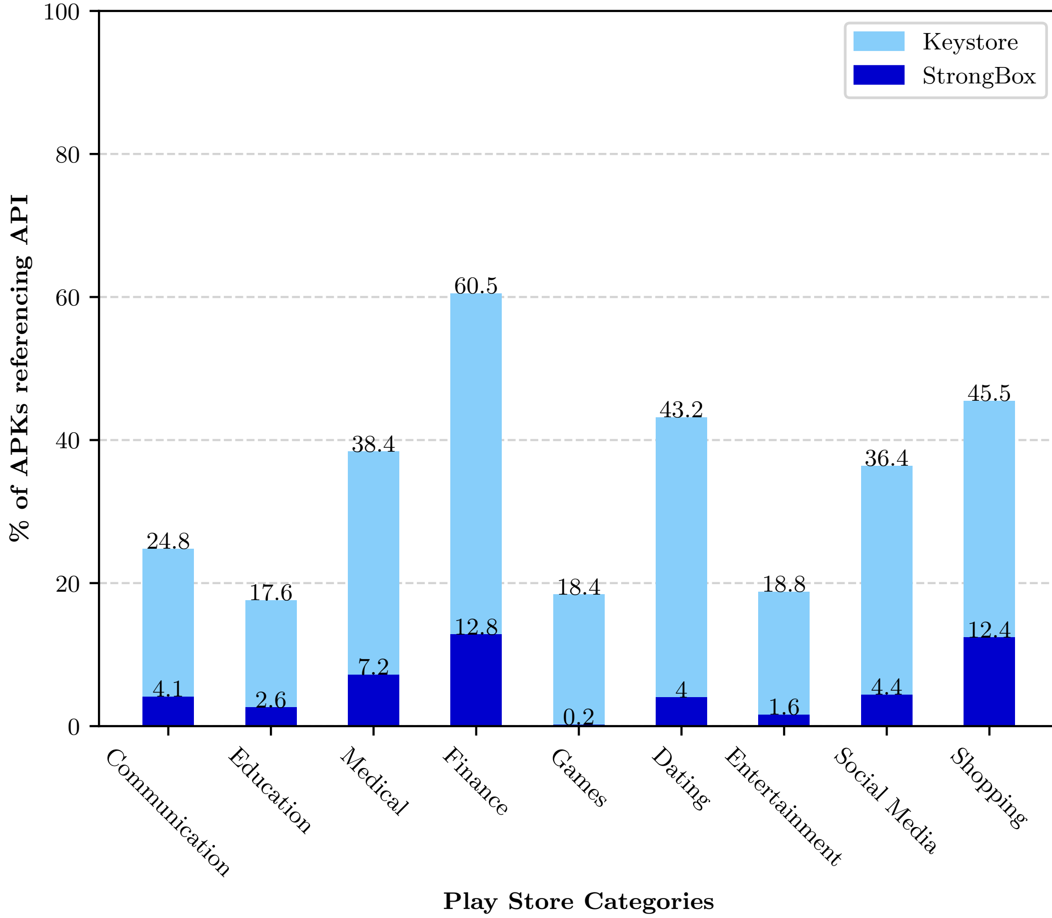

Figure 4.2: Percentages of Android apps using TEE and SE APIs, respectively, across major categories within the Google Play Store. StrongBox usage is shown here as a subset of Android Keystore API usage (i.e., any app that uses StrongBox necessarily uses the Android Keystore API).

distinguish between first-party (e.g., unique) calls and third-party library calls. Similarly, we will frequently distinguish between API usage as a percentage of total *calls* for a particular API method and percentage of individual *apps* containing at least one reference to the method in question since a single app can reference the same method numerous times.

4.3.1 Overall Usage

Of the 486,234 in our dataset (apps currently in the Play Store with at least 10,000 downloads) which we were able to download and decompile, through keyword searching as described in §4.2.3 we find 122,305 apps containing a reference to the Android Keystore API within their source code. This provides us with an upper bound of 25.15% of apps within the Play Store using device trusted hardware. If we consider only the 159,241 apps self-reporting to the Play Store as collecting sensitive data, we find 69,583 apps referencing

the Android Keystore API and the upper bound of trusted hardware use rises to 43.7%.

In practice, these calls may be located within components of third-party libraries not referenced by the app, or within unreachable or legacy source code of the app itself. We then run our in-depth static analysis tool, KeyDroid, on all 122,305 apps flagged as directly referencing the Android Keystore API in some capacity to verify which calls are reachable and collect detailed statistics on how the API is used[2]. We are able to successfully analyze 116,555 apps, with the remaining 2.82% erroring out for various miscellaneous reasons, most commonly exceeding the time limit.

The Android Keystore API further requires developers to specify an intended purpose at the time of key initialization and enforces this purpose when developers attempt to use the key (e.g., a key specified as being intended for encryption cannot later be used to sign). We find that of the 278,056 total init calls for which we were able to retrieve the purpose value, 92.31% of keys are designated as being used for encryption and decryption only, while 5.60% are used for signing or verifying message authentication codes.

A full list of all Keystore API endpoints and their total usage counts is shown in Section 4.2.5. We discuss most methods in more detail throughout this section. We further describe how usage varies by Play Store category in Section 4.3.2, and describe alternative keystores used from a manual review of a subset of apps flagged as not using the Android Keystore API in Section 8.1.

4.3.1.1 StrongBox Usage

We find that 22,875 of the 116,555 apps with any reference to the Android Keystore API (19.62%) further contain a reference to the StrongBox API `setIsStrongBoxBacked(boolean)`, which is 4.7% as a percentage of the overall dataset (and 5.03% as a percentage of apps collecting sensitive data). However, since the API takes in a boolean parameter some of these instances may explicitly request *not* to use StrongBox. To calculate how many apps *enable* StrongBox, we are able to retrieve the argument value for 21,022 out of 24,630 calls and find that while 94.85% of these instances request to use StrongBox, the remaining 5.15% explicitly opt out of using StrongBox and storing cryptographic key material in the device's secure element. Applying this percentage to the 22,875 apps referencing the API, we estimate that 22,367 apps, or 4.6% of our overall dataset, request to use StrongBox for at least one key. This percentage rises slightly to 5.03% if we consider only apps self-reporting collecting sensitive data. To better understand the context behind

[2]A small number of APKs (2,365, or 0.48% of our overall dataset) were flagged as containing the string "AndroidKeystore" but did not contain any references to the actual `android.security.keystore` API when searching the source code. After manual investigation we hypothesize that in most cases this is due to requesting the Android Keystore provider via a different Java API in potentially unreachable code (and so the Keystore API references along the call chain were removed at compilation). We include these APKs in our upper bound percentages reported above but exclude them from more in-depth analysis

these choices without being hampered by source code obfuscation, we manually searched for instances of StrongBox disabling on GitHub [183] as of January 2025. Of the 14 unique (i.e., non-fork) repositories which contained a call disabling StrongBox, two repositories included a comment citing performance reasons while 10 opted out without explanation. The Salesforce Android SDK, for instance, disables StrongBox as the runtime is "too slow" and therefore "not a good fit for [their] use case" [373, 374]. The remaining two instances disabled only if a `StrongBoxUnavailableException` was thrown and were therefore false positives.

The nested structure of Android key generation makes it difficult to reliably link a key generation call (which specifies the algorithm to be used) with the Android Keystore's parameter specification using call objects, and simply checking whether both calls are located in the same method is too imprecise since a single method may generate multiple keys. Instead, we can indirectly estimate ciphers used for StrongBox specifically by linking key size with Strongbox usage. For the 98 keys which set both StrongBox and the key size and for which we are able to retrieve both parameter values, we find that 97 of 98 keys used StrongBox with an `AES-256` cipher while just one key used StrongBox to generate an `RSA-2048 key`, a distribution which again suggests runtime is a major consideration when using StrongBox.

4.3.2 Usage by Category

Figure 4.2 shows the comparative usage of the Android Keystore (TEE) and StrongBox (SE) APIs across a range of categories in the Google Play Store. Unsurprisingly, we find that financial apps demonstrate the highest rates of trusted hardware usage, with over 60% of apps referencing the broader Android Keystore API and 12.8% referencing the StrongBox API. Gaming apps have an exceptionally low rate of StrongBox usage, which we hypothesize is due to the fact that the vast majority of StrongBox references come from third-party APIs, but gaming app development teams are less likely to use high-level app development toolkits given the more advanced functionality required to create the app.

4.3.3 First-Party vs. Third-Party Usage

Here we present a package-level analysis of the location context in which trusted hardware is referenced. In particular, we are interested in determining whether apps flagged as using trusted hardware are doing so as part of the core application source-code or because the hardware API is referenced indirectly as part of a third-party library. First-party usage indicates that developers have consciously chosen to store cryptographic key material in trusted hardware, while for certain third-party libraries (such as analytics libraries) developers may be unaware that this is occurring.

Package Name	Call Count
com.google.android.gms.internal .firebase-auth-api	30,055
androidx.security.crypto	26,345
com.appsflyer	23,566
androidx.biometric	15,960
com.microsoft.appcenter.utils.crypto	12,282
com.google.crypto.tink.integration.android	11,656
com.flurry.sdk	7,806
com.amazonaws.internal.keyvaluestore	4,138
com.oblador.keychain.cipherStorage	4,073
com.huawei.secure.android.common .encrypt .keystore.aes	2,794

Table 4.2: Top 10 third-party libraries referencing the Android Keystore key initialization API `<init>(java.lang.String, int)`. We chose to classify `androidx.security.crypto` [52] as third party after finding that the majority of references were to the `EncryptedFile` and `EncryptedSharedPreferences` classes which abstract the details of key generation and storage.

Overall, we find that the vast majority of Keystore API usages are located in third-party source code (definition provided in §4.2.3.1). Of a total of 199,156 calls to the Keystore init method located in non-obfuscated packages, we find that 94.69% of calls originated in third-party libraries, while 5.31% are located in first-party source code. This observed distribution is also true for SE usage. Of the 17,400 StrongBox calls located in non-obfuscated packages, 98.31% are located in third-party libraries, while only 294 (1.69%) are first-party calls within custom app source code.

4.3.3.1 Third-Party Libraries

A natural follow-on question is *which* third-party libraries referencing the trusted hardware API are most commonly used by apps. Table 4.2 shows the top 10 third-party libraries used by Android apps to reference the Keystore API. While several of the top 10 are security-focused libraries, four are primarily app development and analytics libraries, suggesting that the details of key generation and storage are abstracted from developers who may be unaware of what data is stored where.

4.3.4 Key Authentication

The Android Keystore API allows for a variety of authentication configurations to determine when a key can be accessed. The core authentication method `setUserAuthentication-Required(boolean)` requires users to authenticate via any available form of device unlock

Package Name	Call Count
androidx.security.crypto	11,424
com.oblador.keychain.cipherStorage	2,161
com.microsoft.identity.common.internal .platform	1,019
com.salesforce.marketingcloud.sfmcsdk .components.encryption	758
com.iproov.sdk.crypto	179
androidx.tracing	136
com.ionicframework.IdentityVault	129
com.oblador.keychain.g	108
com.epicshaggy.biometric	76
com.it_nomads.fluttersecurestorage.ciphers	66

Table 4.3: Top 10 third-party libraries referencing Android's secure element StrongBox Keymaster API. We chose to classify `androidx.security.crypto` [52] as a third-party library after finding that the majority of references were to the `EncryptedFile` and `EncryptedSharedPreferences` classes which abstract the details of key generation and storage from the developer.

(device pattern/PIN/password or biometric credentials) for any cryptographic operations using a private key [48]. More specialized API methods allow developers to require a specific form of authentication (e.g., biometric authentication only) and to set the duration during which the authentication is valid.

We find that 15.84% of keys stored in the Android Keystore require some form of user authentication prior to granting access, with 2.78% requiring biometric authentication specifically (and disallowing any other form of authentication, such as device passcode).

By default, if a key requires any form of authentication then a user must authenticate each time the key is used. To provide a more user-friendly configuration, the Keystore API allows developers to set a validity duration period in seconds during which the key can be reused without any need to reauthenticate. 21.75% of keys require the user to authenticate each time they initiate an operation requiring key access, the most secure configuration but also one that can use friction for the user experience. For calls that set a specific duration, the most popular durations were 5 seconds (set by 38.53% of keys which set a duration) and 1 hour (set by 4.45% of keys). A significant percentage of API calls set very short validity durations: 13.2% of calls that set a duration set it to 3 seconds or less, meaning that the user can only reuse the key within the next few seconds. For some use cases, unless the user proceeds very quickly this is effectively the same as requiring authentication each time.

As an alternative to requiring a user to provide information to authenticate, a user can instead approve a pop-up message via the `setUserConfirmationRequired(boolean)` API

before proceeding. As a standalone API this does not require the individual approving the message to provide any information indicating that they are indeed the device owner (i.e., they need only tap to approve), but it can be used in combination with the authentication APIs described above to provide cryptographic certification that a user has approved a certain action. However, we find that very little use of this feature: of the 26 calls to the `setUserConfirmationRequired` API detected where we were able to retrieve the argument value, only two of them enabled confirmation (with the remaining 24 disabling).

4.3.5 Implementation Security

The first step in Android key generation is to create an object of a key generator class. Android provides a variety of key generator APIs, including `javax.crypto.KeyGenerator`, `javax.crypto.Cipher`, and `java.security.KeyStore`. All follow the same paradigm of requiring the developer to provide a string indicating the algorithm used to generate the key as an initial argument (e.g., "AES") and optionally a second argument indicating which keystore provider (e.g., AndroidKeystore, Bouncy Castle, etc.) is responsible for generating the key.

Cipher	Usage Count
AES	147,529
RSA	80,096
HMAC-SHA256	2,321
EC	2,233
HMAC-SHA512	100

Table 4.4: List of ciphers requested for the Android Keystore provider through the javax.crypto.KeyGenerator, java.security.KeyPairGenerator, javax.crypto.Cipher, and java.security.KeyStore APIs along with respective usage counts. We include results from both the "AndroidKeyStore" and "AndroidKeyStoreBCWorkaround" providers due to an Android bug dating back to 2015 [39].

4.3.5.1 Ciphers

Of 232,283 key generation calls to Android cryptographic APIs requesting the Android Keystore as provider, 63.51% requested an AES key, 34.48% requested an RSA key pair, and 0.9% of keys requested an EC key pair. Table 4.4 shows the full list of requested ciphers and their respective usage counts. The low prevalence of EC keys compared to RSA keys is likely due in part to the Android Keystore only supporting asymmetric encryption or decryption using RSA (and so EC key pairs can only be used for signing).

As a point of comparison, of the 20,042 calls requesting the AndroidOpenSSL software-backed provider, 99.74% generated an RSA key pair with just 51 generating an AES key. We hypothesize that developers avoid hardware-backed key storage for asymmetric encryption out of performance concerns, which we discuss further in subsequent sections.

The Android Keystore API also includes legacy ciphers for backwards compatibility and interoperability, some of which have since been deprecated. 3DES, for instance, was simultaneously added and deprecated in API level 28 [56]. In our analysis, we fortunately find very few instances of apps using insecure or legacy ciphers. In particular, we find no instances of `3DES` or `HMAC-SHA1` even though these ciphers are available within the Android Keystore [57].

4.3.5.2 Defaults

Android Keystore API defaults are significantly more secure than those of software-based Java cryptography APIs historically available in Android. For instance, if a developer requests an AES cipher without specifying the accompanying encryption mode(s) as in `javax.crypto.Cipher.getInstance("AES")`, Java's Cipher API defaults to AES with ECB mode, an insufficiently random configuration [40].

Android Keystore, on the other hand, disallows various insecure cryptographic operations by default, including using ECB mode in symmetric encryption, RSA encryption/decryption without proper padding, and using an insufficiently random IV [47]. All of the six essential rules in cryptography laid out by Egele et al. [139] (e.g., do not use ECB mode with symmetric encryption, do not use a non-random IV for CBC) in 2013 are not possible within the Keystore API by default. Unless the developer explicitly disallows randomized encryption, many of the same configurations that run smoothly or are even the default in Java's software APIs will throw an `InvalidKeyException` with the Android Keystore. In addition to disallowing insecure configurations by default, the Android Keystore API is designed such that it requires developers to provide specific configurations instead of providing only a high level cipher (e.g., for symmetric encryption a developer must specify the block mode(s) and encryption padding at the point of key generation using the designated `setBlockModes` and `setEncryptionPaddings` APIS [61, 62]). Android Keystore then verifies that the configuration provided is valid, sufficiently secure, and compatible with the specified key purpose.

4.3.5.3 Randomized Encryption

It is possible, however, for developers to circumvent Android Keystore's secure default settings and implement known insecure configurations by setting Keystore's `setRandomizedEncryption(boolean)` API [47], which mandates configurations must be sufficiently randomized to provide indistinguishability between ciphertexts given chosen plaintexts (e.g.,

IND-CPA), to false. In general, disabling this API means that the same plaintext encrypted with the generated key may produce similar or identical ciphertexts.

We find that 77.94% of calls to the randomized encryption API disable the setting (a relatively unsurprising result given that it is enabled by default, and so referencing the API with `True` as the argument has no effect). When estimating how this configuration is distributed as a percentage of all hardware-backed keys, however, given that there were 30,245 references to the randomized encryption API endpoint we estimate that approximately 8.45% of Android Keystore-backed keys across all analyzed apps disable randomized encryption, a surprisingly high percentage given that this configuration violates a core cryptographic security property.

There are a handful of scenarios in which a developer may deem it necessary to disable this requirement (for instance, if a custom IV is needed), though the API documentation suggests alternative workarounds to avoid disabling randomized encryption for several common cases [47]. To investigate this further, we identify the ten most-used libraries containing a call disabling randomized encryption and manually review each, though we find only three are public without significant reverse engineering: (1) `com.amazonaws.internal.keyvaluestore`, AWS's internal keystore which generates a secure key configuration (`AES/GCM/NoPadding`) but disables randomized encryption because the API "does not work consistently in API levels 23-28" [78], (2) `com.apptentive.android.sdk.encryption.resolvers`, a customer engagement platform which uses a custom initial vector (IV) and thus is required to disable randomized encryption [75], and (3) `dev.mcodex.RNSensitiveInfo`, a React Native wrapper library which disables randomized encryption for a basic AES/GCM/NoPadding configuration as AWS did [308]. Our results are inconclusive as we manually searched Android bug trackers for historical issues with randomized encryption API and could not find any relevant results, but these reported issues with consistency may be an area for the Android team to issue public guidance.

4.3.5.4 Key Attestation

Android Keystore allows developers to require key attestation, which verifies that keys are indeed stored in device hardware [46]. We find 2,724 calls to `setAttestationChallenge(byte[])`, indicating that 0.98% of keys across analyzed apps generate an attestation certificate chain. While still a relatively small percentage, this nonetheless represents a significant increase from Imran et al. [218] who previously scanned a randomly sampled subset of 112,886 Android apps for attestation in January 2021 and found only 5 apps using key attestation.

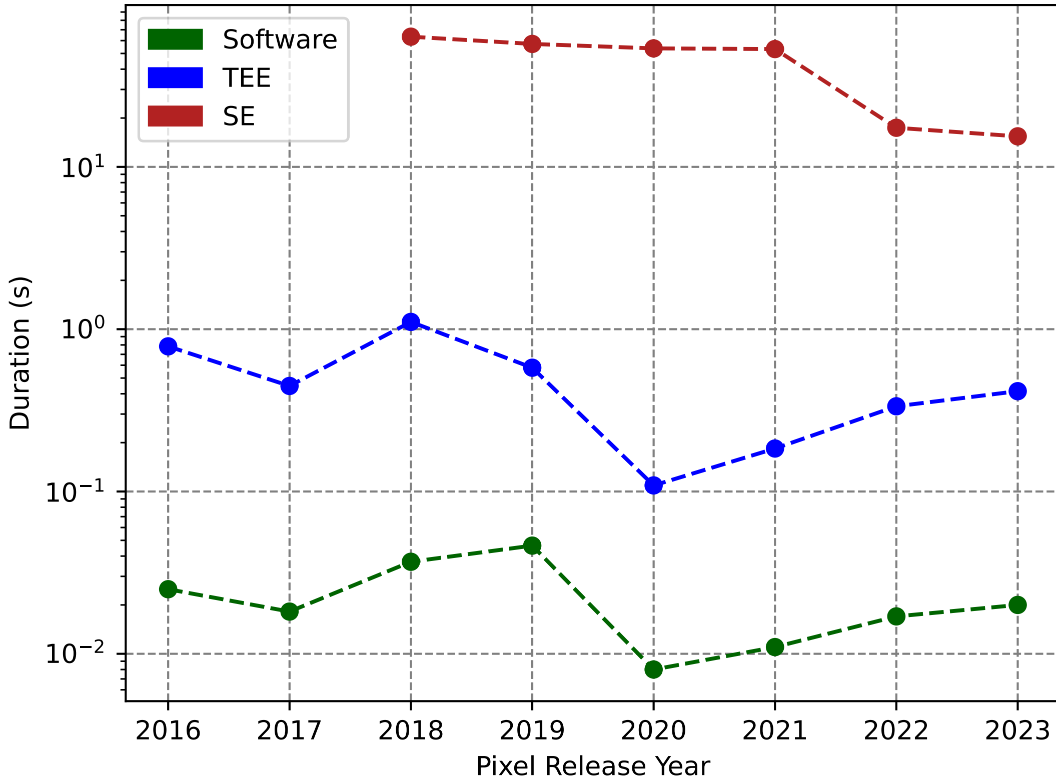

Figure 4.3: Performance evolution of encrypting 1 MiB with AES-GCM in Pixel devices. Each data point corresponds to the Pixel device released in that year (e.g., 2023 represents measurements taken from the Pixel 8). The y-axis is log-scaled.

4.4 Key Storage Performance

Having surveyed the current usage of trusted hardware, in order to judge when hardware-backed key storage *should* be used we must first understand what performance differences, if any, exist compared with software-backed key storage. Unfortunately, to the best of our knowledge Android does not currently publish any empirical statistics evaluating the performance of software or hardware keystores.

To conduct our own measurements, we use AWS Device Farm [6] to measure the runtime performance of key storage options across a variety of Android devices. Our test app calculates the runtime performance of each individual operation for the following three keystores: the device's default software-based keystore (Bouncy Castle for the Pixel XL and AndroidOpenSSL for all other devices), the Android Keystore using the default TEE configuration, and the Android Keystore using a SE (StrongBox Keymaster). The numbers reported below for each operation represent the average performance across 100 distinct iterations.

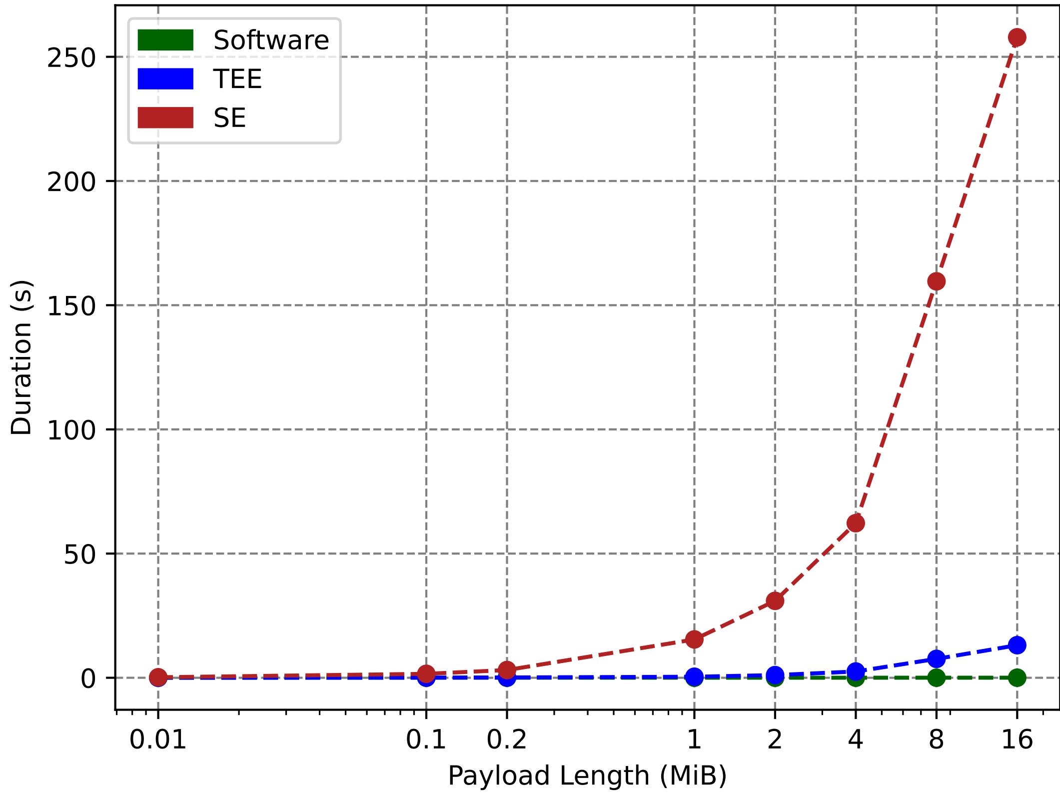

Figure 4.4: Execution times of AES-GCM-256 encryption as a function of message length on the Pixel 8. The x-axis is log-scaled. The corresponding precise numerical runtimes are shown in Table 4.5.

4.4.1 Performance Evolution

We first measure how key generation and encryption performance has changed over time using Google's flagship Pixel device line from 2016 through 2023.

4.4.1.1 Key Generation

Our results show that symmetric key generation has a negligible performance impact regardless of the keystore used. In the most recently released Pixel device, the Pixel 8, generating an AES-GCM-256 key takes 0.002 s in Android's software keystore, 0.021 s in Android's TEE keystore, and 0.071 s in Android's SE keystore, StrongBox. We observe similar runtimes for older Pixel devices. While this represents a large percentage difference, the real runtime impact is negligible given the small execution times. Runtime differences are more significant with asymmetric encryption: in the Pixel 8, generating an RSA-2048 key takes 0.21 s in a software keystore, 1.93 s in Android's TEE keystore, and 9.22 s in StrongBox.

4.4.1.2 Encryption

Figure 4.3 shows the comparative performance of encrypting a randomly generated 1MiB payload with AES-GCM-256 with no padding across Pixel devices released between 2016 and 2023. Android introduced a secure processor beginning with the Pixel 3, and consequently StrongBox measurements are only shown from 2018 on.

The performance impact of software-backed encryption and TEE-backed encryption has roughly stayed the same over time, with the original Pixel and the most recent Pixel 8 reporting TEE measurements of 0.78 and 0.41 seconds respectively. For a payload of 1MiB or smaller there is minimal difference between running cryptographic operations inside a TEE and running them natively in terms of what is observable to the end user (i.e. a runtime of 1 second versus 0.1 seconds will not be noticeable to most users). This has been the case since the initial release of the Android Keystore API.

Message Size (MiB)	**Avg. Runtime (s)**	
	TEE	SE
0.01	0.03 ± 0.01	0.21 ± 0.01
0.1	0.08 ± 0.01	1.59 ± 0.02
0.2	0.12 ± 0.02	3.11 ± 0.02
1	0.42 ± 0.06	15.43 ± 0.10
2	1.13 ± 0.09	30.88 ± 0.16
4	2.56 ± 0.19	62.23 ± 0.33
6	4.25 ± 0.74	94.68 ± 0.37
8	7.67 ± 1.02	159.61 ± 0.72
10	5.83 ± 0.63	127.01 ± 0.69
12	9.24 ± 0.83	192.37 ± 0.80
14	10.87 ± 1.14	223.44 ± 1.02
16	13.10 ± 1.44	257.69 ± 1.09

Table 4.5: Execution times of AES-GCM-256 encryption as a function of message length. These measurements were taken from the TEE and SE in Google's Pixel 8 device.

StrongBox encryption, however, is significantly slower than the other two keystore types. In the Pixel 8, for a 1 MiB payload StrongBox symmetric encryption takes an average of 15.43 s while TEE encryption takes 0.42 s and encryption using a software-backed key takes just 0.02 s. StrongBox performance has improved over time, and so execution times are even longer in older devices: the initial Pixel 3 (released in 2018) has a symmetric encryption runtime of 63.43 s which held reasonably steady until the release of the Pixel 7

in 2022 where the performance dropped significantly to 17.42 s. The sharp performance improvement between the Pixel 6 and Pixel 7 is somewhat surprising since both devices use Google's in-house Titan M2 security chip [257]. The Pixel's main processor changed from Google Tensor in the Pixel 6 to Google Tensor G2 between the 6 and 7 devices, however, and it is possible that the main Tensor G2 processor is able to communicate with the Titan M2 chip more efficiently.

For asymmetric encryption using the same RSA-2048 cipher as above, we measure Pixel 8 performance across keystores on a very small payload of 256 bits (i.e., the use case where a software-backed AES key is encrypted by a hardware-backed RSA key). We find that asymmetric encryption and decryption incurs very little performance overhead on minuscule payloads regardless of keystore, with TEE encryption taking an average of 0.0065 s and StrongBox encryption taking 0.0125 s on average.

Overall, symmmetric encryption using a SE-backed key is roughly 35 to 55 times slower than encryption using a TEE-backed key depending on the device, likely due to either the cost of round-trip communications between the main processor and secure processor, or to a simpler processor architecture overall. This finding somewhat contradicts Android's official documentation, which qualitatively describes StrongBox as "a little slower" as previously mentioned in §2.2.3. On the most recently released Pixel device, however, basic symmetric encryption of a 1MiB payload within StrongBox takes around 37 times (and 15 seconds) longer than the same operation within a TEE. Even so, these runtimes do not indicate that SEs should be avoided, but rather that SEs are only sufficiently performant when encrypting a very small payload (such as a software-backed encryption key).

4.4.2 Performance vs. Payload Length

We further measure the impact of message length on encryption performance. Figure 4.4 shows the performance of payload sizes between 1MiB and 16MiB for the Pixel 8 (again using AES-GCM-256 with no padding). In this experiment we used the average of 10 iterations for payloads 4MiB and above (instead of 100 iterations as with other experiments) due to rapidly increasing execution times.

Message Size (MiB)	Avg. Runtime (s)	
	TEE	SE
0.01	0.02 ± 0.01	0.15 ± 0.02
0.1	0.06 ± 0.01	0.99 ± 0.07
0.2	0.14 ± 0.02	1.89 ± 0.02
1	0.48 ± 0.03	9.05 ± 0.05
2	0.89 ± 0.04	17.99 ± 0.06
4	1.76 ± 0.05	35.91 ± 0.09

Table 4.6: Execution times of generating an Elliptic Curve Digital Signature Algorithm (ECDSA) signature with SHA-256. These measurements were taken from the TEE and SE in Google's Pixel 8 device.

While encryption runtime increases linearly with message length for all three keystore types, StrongBox runtime quickly becomes unmanageable for large lengths. A relatively small payload of 0.1 MiB takes the Android Keystore 0.08 s to encrypt using the TEE and takes StrongBox 1.59 s. A 4MiB payload, however, will take StrongBox roughly 1 minute to encrypt, while the TEE-backed keystore can encrypt the same payload in just 2.56 s, making the TEE viable even for larger message lengths. A software-backed keystore provides the best performance by far as expected, encrypting payloads of up to 16 MiB in just 0.3 seconds given that all operations are in-process with no IPC calls or context switch. Table 4.5 contains the TEE and SE execution times and standard deviations for all message sizes tested on the Pixel 8 (shown visually in Figure 4.4).

Execution times for message signing are similarly cost-prohibitive using StrongBox. As shown in Table 4.6, while StrongBox needs only 1 second to sign a small payload of 0.1 MiB, this runtime increases to 9 seconds for a payload of 1 MiB and 35.91 seconds for a 4 MiB payload. In contrast, a TEE is able to sign a 4 MiB message in 1.76 seconds, making it roughly `20x` faster than StrongBox.

4.4.3 Cross-Provider Performance

We further investigate how Pixel performance compares with other commonly used mobile devices in the Android ecosystem. Figure 4.5 shows TEE performance for symmetric encryption across a range of Android devices, including Samsung and Xiaomi. The five devices measured were chosen by selecting the most recently released device across all device lines available through AWS Device Farm. Four of the five devices measured (Samsung Galaxy A15, Samsung Galaxy A35, Samsung Galaxy S24, and Xiaomi 13)

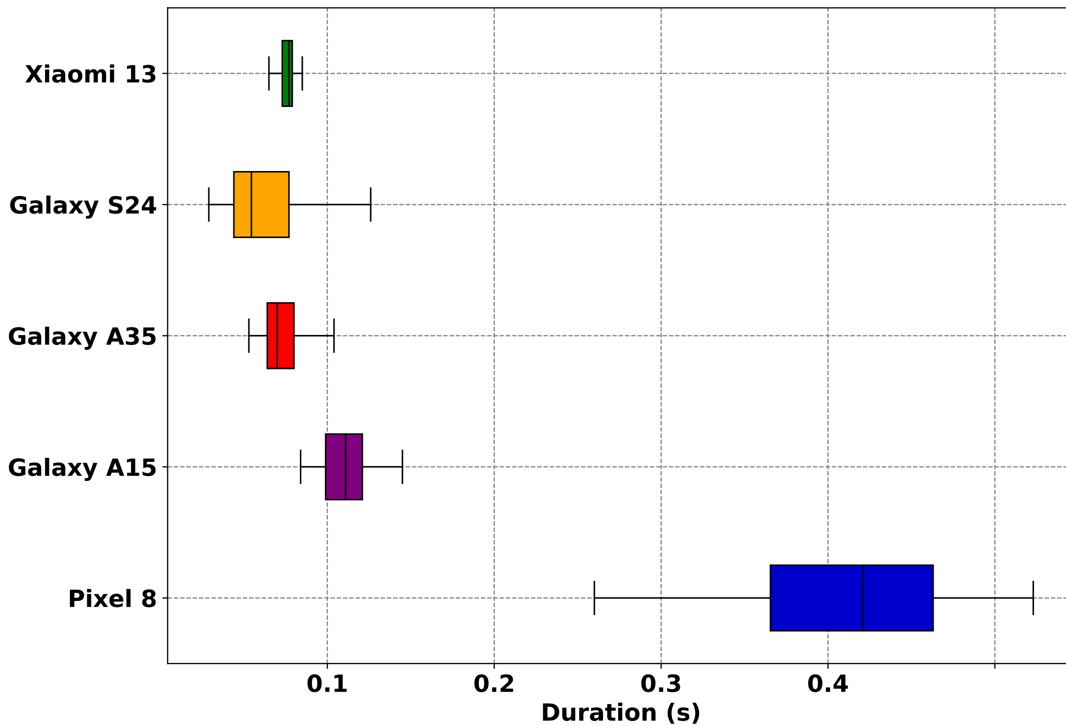

Figure 4.5: Runtime duration of encrypting 1 MiB with AES-GCM within a TEE across a range of Android devices recently released in the past two years. While four of the five devices cluster around 0.1 seconds, the runtime of the Pixel 8 is noticeably longer and with a wider range.

consistently report runtimes around 0.1 seconds for TEE-backed symmetric encryption, while the Pixel 8's average runtime is 0.41 seconds.

While the Galaxy A15, Galaxy A35, and Xiaomi 13 devices do not include a secure element, [3] we compare StrongBox performance between the Pixel 8 and the Samsung Galaxy S24 (released in January 2024) and find a noticeable difference in performance in symmetric encryption. As previously discussed above the Pixel 8 takes 15.43 s to execute AES-GCM for a 1MiB payload, while the Galaxy S24 takes 26.39 s, or close to twice as long. Curiously, the inverse is true for these two devices when considering TEE performance as shown in Figure 4.5: the Pixel 8 takes 0.41 s to execute symmetric encryption using a TEE-backed key, while the Galaxy S24 takes far less time at 0.06 s, illustrating the nuances and complexities of each individual device's processor(s) and other hardware.

[3]Samsung first introduced a secure element in 2020 but only within its Galaxy S series [328]. Devices recently released as part of other series (such as the Galaxy A15 and Galaxy A35 devices, introduced in December 2023 and March 2024, respectively) do not include a secure processor (and thus throw a `StrongBoxUnavailableException` if a developer attempts requests to store keys in the StrongBox). We confirmed this through our own tests.

4.5 Developer Survey

To better understand why Android developers opt not to use hardware APIs, we conducted a developer survey in August 2024 for apps flagged as matching either of two trusted hardware configurations of interest. This study was approved by our department's ethics committee (equivalent to IRB), and all response data was aggregated and anonymized. The survey questions are given in §8.2.

We are interested in two broad categories of apps and conducted separate surveys for each: (1) `Sensitive-NonKeystore`: apps that self-reported as collecting sensitive user data but did not use Android's trusted hardware APIs (either in first-party *or* third-party components) and (2) `StrongBox-Disabled`: apps that referenced the Android Keystore API in a first-party context but explicitly disabled StrongBox for at least one key (e.g., they requested to only use TEE-backed key storage). Both of these high-level configurations indicated that the app developers may have made a conscious decision not to use some form of trusted hardware.

For the first (`Sensitive-NonKeystore`) configuration, we surveyed a random sample of 10,000 developers via email using the contact information given on the Play Store, and have received $n = 42$ responses at the time of writing. We attribute the low response rate in large part to the use of Play Store app support email addresses, which are often read by a customer service team (if one exists) and who may not pass on our survey request to developers.

Of the 42 responses, 18 respondents reported that one factor in opting not to use trusted hardware APIs is that their app does not store credentials and/or deemed the security benefits unnecessary given the type of user data collected. Three respondents reported general performance concerns, while 14 respondents indicated that legacy development or compatibility reasons were prohibitive factors, reporting either a desire to maximize devices the app can run on or that the app was developed prior to the Android Keystore API release date in 2013. One such developer specified that their app uses SQLite due to "lack of knowledge [of the Android Keystore API] at the time of development and difficulties for migrating later." Notably, API usability did not appear to be a widespread concern—just two of the 42 respondents indicated they had found the Keystore API difficult to use.

For the `StrongBox-Disabled` configuration, after filtering out third-party StrongBox calls we identified $n = 25$ apps matching a StrongBox-disabled configuration. Unfortunately we received no responses for our `StrongBox-Disabled` survey, a relatively unsurprising response rate given our restriction of the dataset to first-party disabled calls limited our sample size. Even so, our manual review of disabled StrongBox configurations on GitHub described in §4.3.1 has also provided a window into developers' thought processes.

4.6 Summary

Overall, trusted hardware usage is still comparatively low. While both industry and government have launched various initiatives encouraging developers to move towards trusted hardware [7, 1, 422], usage remains stubbornly low even among apps that, by their own admission, collect potentially sensitive user data. Just 43.7% of apps processing sensitive data use any form of trusted hardware, and almost all of this usage comes from third-party components. While some of these apps may be collecting relatively benign data (such as a user's name) or may rely primarily on a remote server to handle most cryptographic operations instead of storing data on device, this is still a comparatively low rate given there is little to no performance drawback for common cryptographic use cases in a TEE-backed keystore.

Additionally, the vast majority of apps using hardware-backed storage use a TEE instead of an SE (43.7% compared to 5.03%). Put another way, while Google's public goal is to make the SE the "lowest common denominator" in credential storage [1], as of 2024 we observe that only around 10% of apps using trusted hardware at all are using the SE at least once. As side-channel attacks become ever more sophisticated and effective [106], it is even more important for applications to use the most advanced storage available to protect data.

In addition to the protection secure hardware provides against OS compromise, the Android Keystore API also offers significantly more secure defaults than similar Java cryptographic APIs. Android Keystore mandates an `IND-CPA`-secure configuration by default, disallowing insecure configurations that have plagued other cryptographic APIs [139, 174]. Android also runs checks to ensure the security and validity of a configuration, including cross-referencing the stated purpose of a key with which it is generated (e.g., EC keys cannot be used for encryption and decryption, only signing). While it is still possible for developers to circumvent this default (as 8.45% of them do), this nonetheless requires a conscious decision by the developer. Android Keystore's default settings alone make it a security improvement over other cryptographic APIs.

Perhaps most importantly, there are significant performance differences between different forms of trusted hardware in certain cases. TEE-backed storage performance is generally viable for small-to-medium message sizes: We find a negligible difference (¡0.5 seconds) between TEE-backed and software-backed cryptographic operations for payloads less than 1MiB, empirically confirming that in common scenarios hardware key storage runtime is not a prohibitive factor when using the Android Keystore API. A TEE keystore can thus provide significant security benefits with minimal performance impact, providing an ideal trade-off between enhanced security and performance overhead for most app use cases.

In comparison to the TEE, Android's SE demonstrates significantly worse processing

time for all but the smallest payloads. If we consider acceptable processing times to be less than three seconds, StrongBox can only encrypt message sizes of roughly 0.2 MiB or less even in the most recently released Pixel devices. For comparison, a TEE can encrypt message sizes of up to around 2 MiB within the same time frame. Our performance measurements, static analysis of symmetric versus asymmetric usage patterns, and manual review of calls disabling StrongBox all strongly suggest that performance is a prohibitive factor in using SEs in practice. 5.15% of developers referencing the Android Keystore API explicitly opt out of using StrongBox (as in the Salesforce example in §4.3.1).

Even so, StrongBox's execution time may be entirely reasonable in cases with very small payloads: for instance, an app may use StrongBox to encrypt a different cryptographic key. Equally, developers may evaluate overhead cost differently depending on whether it is a one-time operation (e.g., initial login) or a repeated process. Developers need quantitative information in order to make case-by-case decisions, a gap which our work fills. Most importantly, Android's documentation arguably understates the depth of the performance drawbacks of SEs, making it more challenging for developers to make an informed decision. Updated, empirical performance measurements based on contemporary device measurements should be publicly available and easily accessible to developers in place of the ambiguous language currently used in the documentation, which may also have led developers to opt out of using StrongBox as a precautionary measure.

Chapter 5

Keys Not Under Doormats: Recovery in End-to-End Encrypted Systems

A major decision choice in E2EE systems is which authentication and recovery schemes are available to a user, particularly as more and more services are deploying E2EE. The combination of E2EE credential storage and E2EE messaging, email, and cloud storage services represents a rapid shift towards deploying E2EE within web authentication and mobile apps. While some E2EE services are only adopted by a niche segment of users, passwordless authentication and accompanying password managers are targeted at the general public. Though an individual website may not adopt E2EE in any capacity, the credentials of a growing percentage of users will be E2EE, raising the prospect of inadvertent loss of access. This development raises concerns around data recoverability since, by definition, service providers cannot access user data in an E2EE service. Therefore users must take an active role in maintaining access to their account, often including the responsibility for storing their decryption keys.

End-user usability has been one of the most significant barriers to large-scale adoption of E2EE for long-term storage, and it is critical to identify specific research gaps and coalesce on standards to increase widespread E2EE adoption. Providers offer a variety of recovery mechanisms to prevent users losing access to their data, some more user-friendly than others, but there is an unavoidable trade-off in user account security: the easier it is for a user to recover their account, the easier it is for an attacker to gain access. At the same time, loss of cloud storage containing years of photo and document storage is a much more concerning prospect than loss of ephemeral communication history for most users, motivating a fresh look at industry deployments. End-user authentication is now at an inflection point, as companies are in the process of deploying novel security and authentication schemes representing a significant departure from existing user experiences.

While E2EE represents a major improvement in the security level offered by credential managers and other E2EE web services, its use raises natural follow-up questions around

recovery and usability. E2EE comes at a price: if only the user has all the information needed to access the data, the service provider is unable to come to the rescue should a user forget their password or lose their client device. Even loss of access to an E2EE password manager is a scenario the user can often indirectly recover from through manual service-by-service email- or SMS-based password resets.

The prospect of E2EE for cloud backup services such as Apple iCloud make security and recoverability trade-offs more salient. Users store troves of important photos, documents, and other valuable long-term data in cloud services. Account loss with a cloud service provider can be devastating: stories of users who had their accounts shut down after being unfairly flagged by Google as uploading inappropriate material [246, 248] vividly illustrate the practical consequences of sudden account loss, with one such user describing that it "felt as if her house had burned down" [247].

In this chapter, we conduct a comprehensive review of account recovery procedures for E2EE web services (including E2EE cloud storage, email, and messaging), finding that authentication methods in this context vary widely. The majority of service providers rely on asking the user to manually store a recovery key even as more usable variations are feasible, a design choice that arguably makes end users less likely to enable E2EE backups and ultimately undermines the very significant privacy enhancement E2EE provides. Some authentication factors that may be overly cumbersome as part of a daily authentication scheme, such as authentication using trusted contacts, are worth revisiting in the context of E2EE recovery.

This chapter is based on the second half of our paper "SoK: Web Authentication and Recovery in the Age of End-to-End Encryption". For this chapter I adapted the text, figures, and plots to fit with the dissertation. I conceptualized and conducted all experiment and was the primary author of the text. Daniel, Ross, and Alastair contributed towards the development of the ideas and their presentation.

5.1 Related Work

Authentication Frameworks: In 2012 Bonneau et al. [100] introduced a seminal framework for evaluating authentication schemes across security, deployability, and usability, concluding at the time that no scheme met as many desired criteria as conventional password-based authentication. While the desirable properties of any authentication mechanism deployed at scale have generally remained the same, both state-of-the-art industry deployments and academic research understanding have evolved dramatically since their survey—device-based credential protocols such as FIDO2 did not yet exist, and industry had only just begun introducing the concept of multi-factor authentication (MFA) [195]. In 2017, Alomar et al. [32] presented a framework for classifying social

authentication schemes and associated attacks, though in practice few of the social authentication schemes are deployed outside of trusted contact recovery. In 2021, Kunke et al. [263] evaluated 12 common account recovery mechanisms within the 2012 framework of Bonneau et al., though at the time the only FIDO2 protocol deployments were hardware token-based and thus they did not consider passwordless authentication.

Prior comparative surveys of authentication schemes were either conducted well before major contemporary shifts in authentication schemes (namely, MFA adoption, device-based authentication, and E2EE authentication) or focused on authentication schemes pre-dating recent trends (as in Lassak et al. [266], a longitudinal study of the usability of email, SMS, recovery questions, and social authentication as fallback mechanisms). To the best of our knowledge no prior work has surveyed authentication and recovery schemes in an end-to-end encrypted context.

Measuring Industry Deployments: A handful of prior studies have surveyed industry deployments of non-E2EE authentication schemes. In a 2021 survey of the 208 most widely used websites that offer account creation, Gavazzi et al. [177] found that only 42.3% of accounts support MFA and approximately 22% appear to support some form of risk-based authentication (e.g., blocking suspicious login attempts based on geolocation). In this chapter, we particularly investigate E2EE web services, where the account that the user is attempting to recover is encrypted such that the provider definitionally can be of no help in recovering the encrypted data, and furthermore the user may no longer have access to their password or other primary authentication mechanism (e.g., mobile device). Holtervennhoff et al. [212] recently conducted a usability survey of users' perceptions and strategies for handling E2EE recovery keys, but no prior work has looked comprehensively at deployed authentication and recovery mechanisms for end-to-end encrypted data, where either the web service or credentials may be E2EE.

5.2 E2EE Recovery Mechanisms

Here, we outline and discuss deployed recovery mechanisms in E2EE schemes, systematizing currently deployed protocols across E2EE services. For the purposes of this section we do not distinguish between authentication and recovery schemes since our goal is to summarize pathways towards account access. From an attacker's perspective, there is no distinction between authentication and recovery. Table 5.1 shows the diverse array of authentication recovery mechanisms used by the 22 most widely used E2EE providers for storage, email, messaging, and password managers. Our discussion focuses on services designed for individual consumer use rather than services targeted at enterprises (Table 1 presents authentication schemes available in the free version of each service and notes schemes available only in premium [non-business] service versions).

	E2EE Platform	Device Keychain	User-Chosen Password	Recovery Code	Third-Party Storage	PIN	Recovery File	Recovery Email	Recovery Contact	Recovery Group
		Authentication and Recovery Mechanism								
Storage	Apple iCloud	●	●	●	○	○	○	○	●	○
	NordLocker	○	●	●	○	○	○	○	○	○
	pCloud	○	●	●	○	○	○	○	○	○
	MEGA	○	●	●	○	○	○	○	○	○
	Tresorit	○	●	○	○	○	○	○	○	○
	Internxt	○	●	●	○	○	○	○	○	○
	Filen	○	●	●	○	○	○	○	○	○
Email	Proton	●	●	●	○	○	●	○	○	○
	PreVeil	●	○	○	○	○	●	○	○	●
	Tutanota	○	●	●	○	○	○	○	○	○
	StartMail	○	●	●	○	○	○	●	○	○
Messaging	FB Messenger	●	●	●	●	●	○	○	○	○
	WhatsApp	●	●	●	●	○	○	○	○	○
	Signal	●	○	○	○	○	○	○	○	○
Credential Managers	LastPass	●	●	●	○	○	○	○	●	○
	Bitwarden	●	●	●	○	○	○	○	●†	○
	Dashlane	●	●	●	○	○	○	○	○	○
	1Password	●	●	●	○	○	●	○	●	○
	NordPass	●	●	●	○	○	●	○	○	○
	Keeper	●	●	●	○	○	○	○	○	○
	Enpass	●	●	○	○	○	○	○	○	○
	Google PM	●	○	○	○	●	○	○	○	○

Table 5.1: Deployed recovery mechanisms for end-to-end encrypted storage, email, and messaging services. The table displays all recovery options offered by each service, but in practice some of these choices may be mutually exclusive. For instance, WhatsApp allows a user to set either a recovery code or a user-chosen recovery password, but not both. For the vast majority of services, the available recovery schemes are designed to be standalone and used in scenarios where a user has lost their password and/or device. In this context, 'device keychain' scheme is defined as authenticating to the client application using the client device's unlock mechanism. We define and discuss social authentication (recovery contact and recovery group) in depth in § 2.3.3.1. †Only available for premium users.

If a user has access to a logged-in client device, recovery is simple. A common recovery mechanism invisible to the user is to automatically save a decryption key to the browser or device's local keychain where it can be accessed upon device unlock, enabling the user's device to serve as an authentication mechanism. WhatsApp, for instance, allows a user to reset their recovery code through the WhatsApp app after authenticating to their device using biometrics or entering the device PIN (which allow the WhatsApp client to access the encryption stored on device) [449]. Apple's Advanced Data Protection E2EE cloud backup scheme offers a similar recovery protocol [66], and Messenger's Labyrinth protocol allows users to send a one-time code to their old device [312].

The challenging scenario is the case where a user has lost access to both the password used to unlock the account in question and, where applicable, all relevant client devices. For instance, perhaps someone has lost their phone, and attempts to restore WhatsApp on a new phone, only to discover they cannot find the decryption key to the WhatsApp backup.

5.2.1 Recovery Codes

We find that recovery codes are the primary backup method used to recover access to encrypted data, with 17 of the 22 providers surveyed offering or mandating this backup method. We described subtle variations in recovery code deployments in Section 2.3.3.2, but these codes are generally arbitrary alphanumeric strings of 24 to 64 digits intended to be non-human-memorable.

Unfortunately, it is all too easy for users to make mistakes that can lead to accidental account lockout, such as taking a screenshot of the recovery code which is then synced to the cloud storage to which the code restores access [212]. Some services (e.g., Keeper [251]) also make setting up a recovery code optional. Users will have the option of enabling the feature at account setup but can proceed without it. An additional consideration is that non-enterprise cloud services are often used by small businesses and community organizations in addition to personal usage, and access credentials may have high turnover in ways that can make it easier for legitimate users to forget or lose access to credentials.

Recent academic research has suggested that recovery code loss is not uncommon in practice: Holtervennhoff et al. [212] investigated how users perceive and store recovery codes in E2EE services, conducting a user survey of 281 users of Tutanota, an E2EE email service, and qualitatively analyzing Reddit support threads for the same service. They found several support threads in which users had lost their account password, 2FA device, and recovery code, and the user survey revealed a small number of users reported that they had not recorded the recovery code at all, believing there was no chance they would lose their passwords. Approximately 12% of users surveyed believed Tutanota could help them regain access in case of recovery code loss, and only 14.8% of users saved the

recovery code in more than one location. These results are derived from the userbase of a privacy-centered email service, which the authors acknowledge "is not representative of email users or privacy-conscious users in general" [212], so we may anticipate user misconceptions to be even higher in a service targeted at a mass audience.

5.2.2 Human-Memorable PINs and Passcodes

To explore more usable solutions, some providers allow users to enter a shorter recovery code (such as a short PIN) or a more memorable recovery string (such as a seed phrase or user-chosen password). To avoid brute-force attacks, these low-entropy codes are then used to generate a secure encryption key using a key derivation function. Meta's Labyrinth design, for instance, allows users to enter a 4-digit PIN and then uses this short PIN to authenticate to a longer, internal pseudorandom recovery code stored in a rate-limited HSM, limiting the number of attempts to 10. Google Password Manager similarly requires users to set a 6-digit PIN to recover access on a new device [188].

If a hardware exploit is able to overcome an HSM provider's rate limits, however, a short string like a PIN can be brute-forced in as little as a few hours [132, 294, 282]. To address this attack vector, Dauterman et al. [132] proposed distributing the decryption keys among multiple HSMs, and in 2024 Signal deployed a key recovery system that distributes trust among multiple types of HSMs from different vendors since an exploit is unlikely to compromise *all* HSM types [123]. Similarly, Juicebox [333], an open-source key recovery protocol created by Signal cofounder Moxie Marlinspike among others in 2023, enables PIN-based recovery using multiple independent cloud HSM providers but is still in the early stages.

Even with rate-limiting, however, PINs may still be easily guessed as users are also liable to choosing easy-to-guess PINs or PINs based on easily discoverable dates such as a birthday [293]. PINs are also potentially compromised via social attacks such as shoulder-surfing [230] and thus rate limiting mitigates but does not prevent even external attackers from compromising a numeric PIN.

Short, numeric PINs are also not a foolproof strategy to counter the fallibility of human memory as some percentage of users will still invariably lose access to the PIN. In one study of Signal PINs, 12% of participants reported "occasionally, frequently, or very frequently" forgetting their PIN [81]. WhatsApp's E2EE backup scheme offers the option of a user-generated password to authenticate to the pseudorandom recovery code instead of directly storing the recovery code, where the low-entropy password is used to generate the proper 64-digit key using an oblivious pseudorandom function. From the user's perspective, this may be simpler to use as they only need to enter the shorter or more human-memorable sequence. Further variations include a "recovery phrase", a long string of between 12 and 24 words that the user can either store or attempt to

memorize (commonly used for cryptocurrency wallets and sometimes referred to as a "brain wallet" [147, 115]). Proton, an end-to-end encrypted email service, uses a 12-word recovery phrase instead of a more conventional pseudorandom recovery code [361]. The length of recovery phrases, though, makes memorizing these phrases an unattractive option for the general public, with the net result that recovery phrases offer little to no benefit compared with a standard pseudorandom string.

5.2.3 Manual Reset

As a final form of fallback, we note that StartMail, an E2EE email service, offers users the ability to request an email reset [393]. To preserve E2EE, on the backend their scheme requires the keys of two separate staff members to jointly recover the user's lost decryption key. While this scheme is not properly E2EE in the sense that colluding employees can access user keys, it nonetheless presents an interesting case study.

We further observed that every credential manager offering a business or enterprise version offers the ability for an administrator of the business account to manually reset a user's master password even when all other recovery options have been lost [269, 270, 91, 130, 9, 334, 228, 142], though there is some variation in whether this feature is enabled by default or if a business needs to explicitly opt in. This is effectively a variation of trusted contact authentication where the trusted contact is another employee within the organization.

5.3 Takeaways from E2EE Recovery

Having observed that recovery codes are the primary backup method used to recover access to encrypted data, we discuss potential design modifications to improve end-user recovery.

5.3.1 Usability Improvements

There are additional feature choices providers can offer to mitigate the risk that a user loses or forgets their recovery passcode in the first place. WhatsApp and Signal both provide regular password reminders asking users to confirm their backup PIN, though in Signal's case a user optionally has to enter a short PIN while WhatsApp requires users to enter either their password or full 64-digit code before they can access the app [433]. Other providers may offer a similar feature as well, though it is not explicitly stated in the documentation. One study of Signal's opt-in PIN reminder, however, found that roughly quarter of participants in one survey reported that they rarely or never confirmed their PIN when prompted [81]. While mandatory reminders may be more effective, they also

run the risk of becoming a nuisance to users, potentially leading users to disable E2EE storage to avoid these notifications and achieving the opposite of the desired effect.

We also observe patterns of insecure defaults among E2EE recovery. Namely, some services (e.g. Keeper [251]) make setting up a recovery code optional. Users will have the option of enabling the feature at account setup but can proceed without it, a risky architectural design from a usability standpoint as the user now relies only on their master password.

Facebook Messenger's E2EE protocol, Labyrinth [312]), represents an interesting case study of trade-offs between authentication and data recovery. Since Messenger is commonly used as a web application, Labyrinth is designed to allow users to log in on a new device or browser using only their ordinary Facebook credentials. If a user no longer has access to their cryptographic key material (namely, their private authentication key), they will still be able to log in to their account, but will not have access to their conversation history. This is a reasonable trade-off for E2EE messaging, but does not work for E2EE of long-term storage, where the whole purpose is to access previously generated data.

5.3.1.1 Cross-Provider Syncing

A promising feature from a usability standpoint is the ability to automatically store the recovery key in a separate cloud service. Meta's Labyrinth protocol gives users the option to store their pseudorandom recovery code in either Google Drive or iCloud Drive (depending on the mobile platform), where it is stored in a hidden folder in the third-party cloud service and does not preclude the user from separately storing the recovery code elsewhere as well. This form of automated storage represents an improvement from a usability standpoint in that a user would now have to lose access to both cloud providers, but there are privacy considerations: in a study of 2FA backups, Gilsenan et al. [181] found that automatically uploading backups to Google Drive requires the user to grant read access to the additional service for their Google account name, email address, and photo.

Proton provides a slight twist on automatic storage by offering a platform-agnostic encrypted "recovery file" that can be stored long-term and provided to Proton at a later date to restore access [361]. The documentation cryptically suggests that both recovery phrases and/or files "may become outdated", at which point a user will be warned that this recovery mechanism is no longer valid to restore access but will have to generate a new recovery phrase (though it is unclear why or how often this might occur). In general, automatic cross-provider cloud storage represents a real usability improvement over asking the user to store the key on separate physical hardware (e.g. a USB stick) or to write it down on a piece of paper somewhere as both are easy to lose accidentally.

5.3.1.2 Distributing Trust

Distributing trust across multiple providers can mitigate the risks of E2EE account authentication and recovery. HSM rate-limited PIN authentication is among the most usable of authentication and recovery schemes since it only asks the user to remember a much shorter string. Its low entropy makes it difficult to recommend adoption as a recovery scheme at the moment as a hardware exploit would easily compromise E2EE data, but there are currently multiple promising cryptographic proposals or open-source protocols to divide the recovery process among separate vendor cloud services and/or vendor HSMs [123, 333] (e.g., subsets of the recovery code are distributed among different cloud storage services). In principle, distributing trust among two distinct providers is similar to the widely referenced assumption in cryptography of non-colluding servers [432, 436]. Both providers would need to be compromised or collude for data confidentiality to be compromised, and a user would need to lose authentication credentials to both services to become locked out. This is one of the most promising avenues for usable E2EE recovery for the general public.

5.4 Summary

Overall, we find minimal consensus around recovery schemes in end-to-end encrypted systems. While a majority (19) of the 22 E2EE service providers studied either used an arbitrary, alphanumeric recovery code as their primary recovery failsafe or did not provide any recovery mechanism in addition to the master password, beyond recovery codes there is little to no consensus around whether providers *should* offer additional recovery mechanisms and if so, what those mechanisms should be. To improve usability, six of the 22 providers offered the ability to automatically save the recovery code in either third-party storage (e.g. integrated with Google Drive) or as a local recovery file containing the code. Two providers allowed short PINs to be used as a recovery mechanism, while five providers offered some form of social authentication.

Inconsistencies across which authentication and recovery possibilities are provided in the first place is equally as important for accurate user mental models and understanding of recovery possibilities. Prior work has shown that variations in 2FA backup recovery processes make it more likely a user may misconfigure backups [178, 35]. To enable E2EE adoption on a large scale there is a need for industry to standardize available recovery options across E2EE web services, particularly for providers that serve a user base comprised of the general public (e.g., Apple).

We conclude that to prevent lockout, E2EE contexts require a wider set of authentication schemes than general web authentication. Services should offer a variety of recovery methods to cover the diversity of users' personal situations, and offer the ability to enable more than one recovery path for sufficient redundancy. Given that E2EE inherently

lacks provider-assisted recovery as a fallback option, we observe clear differences between authentication schemes deployed in E2EE services compared with schemes deployed in general use cases. In particular, while none of the top-300 websites currently offer recovery via trusted contacts, five of 22 E2EE services offer some form of social authentication recovery.

Chapter 6

Protecting Authentication Credentials Using End-to-End Encryption

Having surveyed account authentication and recovery in E2EE systems in the previous chapter, we conclude by considering the case where the credentials themselves are secured using E2EE. We devote special attention to E2EE credential managers since this is both the most salient example of security-usability trade-offs as well as an actively evolving ecosystem.

Password-based threats have plagued web authentication for as long as the Internet has existed, with Google declaring passwords to be "the single biggest threat to your online security" in 2021 [290]. While industry providers have deployed a wide variety of end-user authentication and recovery schemes in the past decade, developing an authentication scheme that offers a desirable balance of security, privacy, usability, and recoverability for a diverse population of users is an enormous challenge. The inherent difficulty of resolving these trade-offs is the central reason why passwords persist [207, 362, 100]. Even so, industry providers have made significant strides in mitigating password-based threats, such as using browser metadata to detect suspicious logins [450, 176, 317] and offering "single sign-on" options to centralize authentication with a single account.

We are entering a new era: the FIDO2 standard [165] (jointly developed by the World Wide Web Consortium [W3C] and the FIDO Alliance) provides a password-less authentication protocol based on public-key cryptography, commonly referred to as *passkeys*. Passkeys have recently been deployed across all major operating systems and web browsers, and as of May 2024 over 400 million Google accounts have set up a passkey [191]. End-users are able to authenticate to remote web servers using their device authentication (e.g., fingerprint, PIN, etc.) to unlock access to the relevant key material. While passwords as a whole will not disappear any time soon, since they remain the most usable authentication

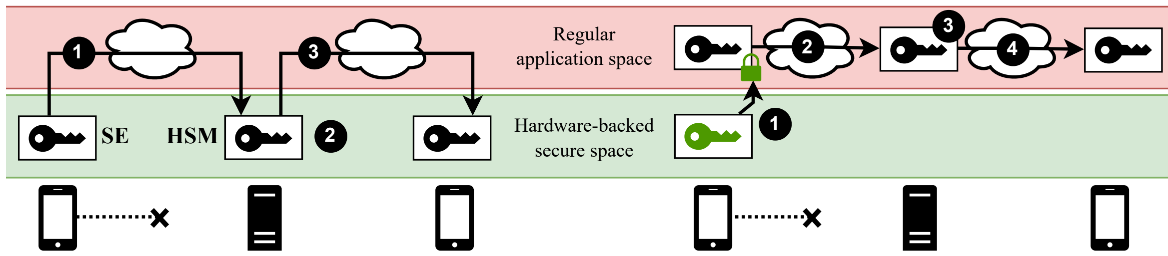

Figure 6.1: Synchronization flows for credentials. *Left:* in ideal first-party syncing, the credentials always live in secure hardware in either a secure enclaves (SE) or hardware security modules (HSM); neither backup (1) nor recovery (3) expose the key to regular application space; the HSM (2) enforces end-to-end authentication and rate-limits. *Right:* third-party syncing more commonly accesses the key material in regular application space; however, it might be encrypted-at-rest (1) using a key inside the SE; it is typically protected in-transit (2, 4) and uses password-derived keys for encryption-at-rest to ensure E2EE (3).

scheme for many, especially at-risk demographics [24, 376], passkeys represent the first genuine contender for password replacement with the backing of major industry players. However, there are still unresolved questions around usability, security, and recoverability, all of which will have a major impact on end-user adoption. To improve usability, users are offered the option to sync device passkeys to either the device's cloud provider or a third-party cloud credential manager, where stored keys are end-to-end encrypted such that they are inaccessible to the provider.

In this chapter, we provide an in-depth discussion of cloud-synced credential managers, which form the foundation of contemporary authentication architectures. All widely used credential managers depend on E2EE to secure a user's authentication credentials even while passing through a third-party server (e.g., a server controlled by the service provider), but there are important security distinctions among the various syncing architectures. We focus particularly on passkeys as a novel authentication credential dependent on E2EE cloud syncing for usability by the general public, and demonstrate that while E2EE cloud sync is essential for usability it nonetheless brings moderate security drawbacks in comparison to non-syncable credentials bound to device hardware. We conclude this section by conducting the first survey of passwordless authentication adoption among the most widely used websites within the U.S. as of May 2024 to better understand which schemes are deployed and where.

This chapter is based on the first half of our paper "SoK: Web Authentication and Recovery in the Age of End-to-End Encryption". For this chapter I adapted the text, figures, and plots to fit with the dissertation. I led the conceptualization, analysis, and was the primary author of the chapter, and my co-author Daniel was the main author of the FIDO2 protocol analysis and developed the accompanying figures. Daniel, Ross, and Alastair contributed towards the development of the ideas and their presentation.

6.1 Authentication Credentials

Contemporary authentication credentials can be assigned to one of two high-level categories: (1) credentials bound to the hardware of a single device and (2) credentials that can be synced across multiple devices. Since device-bound credentials pose significant usability drawbacks (elaborated below in §6.2), the most common paradigm is to store syncable, multi-device credentials in a credential manager (historically referred to as *password managers*, though this is increasingly a misnomer as they are used to store both conventional passwords and cryptographic keys). Credential managers, henceforth *CMs*, are databases of authentication credentials that typically allow synchronization across multiple devices. These are in turn protected by either a master password or external hardware (e.g., YubiKey), reducing the cognitive load for the end-user as they will need to remember at most one password that unlocks all other authentication credentials.

All widely used CMs (see Table 5.1 for a full list of systems considered based on related work and recent rankings of password manager services [254]) depend on E2EE to prevent any party other than the originating user accessing the user's credentials [415]. All data is encrypted on the client device before being synced to the server, typically using keys derived from the master password used to unlock the CM using a key derivation function such as PBKDF2.

In this chapter, we present several important security distinctions between device-bound and syncable credentials, and discuss variations of CM designs which impact the security of syncable credentials. We conclude with an in-depth discussion of *passkeys*, an emerging passwordless authentication scheme that comes in both credential flavors (device-bound and syncable) and illustrates the security and usability benefits and drawbacks of current architectures.

Our understanding and discussion of the security and usability of E2EE synchronization of passkeys relies on the publicly available documentation provided by the vendors. Hence, it does not allow us to verify that the deployed system behaves as described and the published documents naturally cannot cover all details. Where necessary we made conservative assumptions about the provided functionalities drawing on knowledge from similar protocols and implementations.

6.2 Device-Bound Credentials

With the increasing prevalence of hardware tokens and secure hardware chips in smartphones, authentication using only a single strong hardware-backed factor (the user's smartphone) is now viable—allowing providers to remove passwords from daily authentication flows. Most modern devices contain built-in platform authenticators (e.g. Trusted

Platform Module [TPM] in Windows, Secure Enclave in macOS/iOS, and StrongBox in Android).

At a high level, device-bound credentials use public-key cryptography to authenticate, where the smartphone's secure hardware component generates a unique keypair for each web account, stores the private key in the hardware module, and shares the public key with the web server. Importantly, once generated the private key *cannot* be exported from device hardware, and hence it is not possible to backup, sync, or otherwise duplicate the credential. To authenticate to the web service, a user need only authenticate to their local device authenticator using their regular device unlock mechanism (e.g., fingerprint, PIN, pattern).

Most common forms of device-bound authentication are based on the FIDO2 specifications [165] which comprise the WebAuthn standard [455] for client-to-server communication and the client-to-authenticator protocol (CTAP) [163]. FIDO2 was intended from the start as a contender for password replacement [170]. The initial deployed version of the Fast Identity Online (FIDO) protocol in 2019 ("FIDO U2F") relied on two-factor authentication (2FA) hardware tokens (Section 2.3.2.4), and the passwordless standard was released in 2018 [165]. In FIDO terminology, this means there are two types of hardware authenticators a user can use to authenticate: using a *roaming authenticator*, such as a discrete hardware token, or a *platform authenticator*, such as the built-in smartphone authenticator [170].

The primary benefit provided by the FIDO authentication scheme is mitigating phishing attacks and large-scale compromise, since each account has its own distinct keypair. During login, the device signs a challenge to prove possession of the respective private key (see Figure 6.2a). Binding the private key to the original web service ensures that authenticators do not sign challenges coming from malicious websites. Additionally, the graphical interfaces of CMs provide no mechanism to view the underlying cryptographic key, preventing users from sharing the key with an adversary [8].

As a result of these enhanced security properties, single-device credentials are often mandated in enterprise use cases or desirable for highly security-conscious individuals who want to ensure maximal resistance to phishing and other common attacks. The primary downside from a security perspective is that credential security now reduces to device security [170], as a compromised device can allow an adversary access to all services. Smartphone providers have deployed several security measures to prevent unauthorized access, such as requiring biometric authentication each time a passkey is used and instituting device unlock rate-limiting, but the precise security protections will depend on platform and user configurations.

Device-bound credentials suffer from serious usability drawbacks: they inherently mean that a user can only authenticate to a service with the particular device (e.g., laptop

computer) on which the credential is stored. If the device is lost or damaged, the credential is lost and a user may be locked out of their account permanently. This can be mitigated through the use of multiple independent hardware tokens configured to authenticate to the same account, but a user must go out of their way to set this up and may still lose both hardware devices needed to access the account. Creating and storing credentials in a secure hardware device without any option to export the key material is the gold standard from a security perspective, but it falls short from a usability standpoint due to an inherent lack of recoverability or backup options. We discussed recovery schemes in greater detail in Chapter 5.

6.3 E2EE-Synced Credentials

To address the tension between security and usability in device-bound credentials, industry has developed a variety of syncable credential schemes in which credentials can be shared with either a separate client device or a cloud service.

6.3.1 Cloud Backups vs. Routine Device Syncing

Such synchronization can take one of two forms, each with distinct security properties: credentials are backed up and later recovered to a new device; or credentials are continuously synced between two known devices.

6.3.1.1 Backups and recovery

Storing duplicate copies of credential vaults in the cloud is an essential fail safe for device loss. We discuss E2EE recovery options in greater detail in Chapter 5, but cloud-based recovery is generally viewed by providers as an exceptional case and providers have varying processes for regaining access ranging from long-term recovery codes to social authentication. Since this scenario assumes loss of any existing, known devices (and sometimes also the loss of the master password), the cloud backup provider authenticates a login attempt from a new client device based only on the specified device-agnostic recovery mechanism, with no direct key agreement between the client and server.

6.3.1.2 Syncing between known devices

Syncing across active client devices (e.g., propagating an authentication credential newly created on a mobile device to a desktop) is both simpler from a usability standpoint and more secure as it allows for interactive key agreement and authorization that is largely invisible to the user. If a user owns a sufficiently large number of "living devices", such as multiple laptops or a tablet device, each device effectively functions as a backup copy

of the credential vault, making it less likely the user would lose *all* devices and need to initiate a recovery process.

The previous backup and recovery process using a cloud-based HSM can be understood as a two-degree exchange between known devices. In contrast, in this setting the HSM effectively operates as a trusted and attested user device which is being operated remotely. The initial backup step and the subsequent recovery step are therefore just exchanges between trusted devices and the remaining complexity addresses the challenge of remote attestation and authorization.

6.3.2 First-Party vs. Third-Party Credential Syncing.

We further separate CMs into two sub-categories: first-party CMs, where the CM is integrated with the specific device OS (e.g., iCloud Keychain and iOS), and third-party CMs, which are designed to be hosted by any OS (e.g., LastPass).

6.3.2.1 First-Party Credential Syncing

To enhance authentication usability and security, major industry OS providers have implemented their own in-house CMs (e.g., iCloud Keychain). We refer to this as *first-party credential syncing*, or intra-ecosystem syncing, where the cloud backup service and CM vendors are the same. Operating systems with a built-in CM generally sync credentials to their respective cloud backup service by default [188, 70] (iCloud, for instance, automatically adds credentials to the iOS Keychain provided the third-party service developer has specified the credential is syncable [71]).

Importantly from a security standpoint, Apple's iCloud Keychain syncs credentials from one client device enclave to another client device enclave via Apple-owned HSM clusters such that even the encrypted credentials never leave hardware storage during backup or recovery (as shown in the left diagram in Figure 6.1) [69]. Google's Android devices handle public key credentials, e.g. passkeys, similarly, relying on cloud-based HSM devices as well [188]. Because these vendors own the device hardware, device OS (including low-level interfaces), and syncing servers, they are able to design an architecture with significantly greater security guarantees than that of third-party CMs which need to account for every type of hardware and OS [152, 329]. While non-vendor-specific CMs may use secure hardware where available, first-party CMs are *guaranteed* to deploy hardware-backed syncing end-to-end.

6.3.2.2 Third-Party Credential Syncing

The majority of CMs are what we term *third-party* CMs, such as LastPass [271] and BitWarden [90], which can generally be used as either a native application hosted locally

or as a browser extension on most major operating systems and web browsers. One of the most attractive features of these CMs is the ability for credentials to be synced and exported across different ecosystems (i.e., inter-ecosystem syncing).

This flexibility comes with security downsides: in order to be exportable, in most cases [1] the credentials must necessarily enter the application process at some point, leaving them vulnerable to memory extraction attacks [116, 329, 224] and other offline and online attacks on password managers based on adversarial possession of an encrypted database [185, 337, 114], including brute-force attacks to compute the backup encryption key and decrypt the database [84, 135, 131]. These exports (and corresponding imports on another client) happen frequently in most CMs, with services typically synchronizing state after each new credential is added [153]. Of seven widely used third-party CMs analyzed, only one (Keeper) mentions HSMs as part of their server architecture [252], with the others storing the encrypted database vaults directly on their servers [271, 90, 131, 10, 335, 143]. Several CMs provide only high-level details of their server security architecture publicly, so it is difficult to be certain of their design.

6.3.3 Attacks on E2EE Cloud Credential Storage

While E2EE is essential for providing a threshold level of security for credential vaults synced to the cloud, these services still present several security risks not relevant for device-bound credentials. Syncing across multiple devices inherently increases the attack surface, leaving credentials vulnerable to the compromise of the weakest synced device [389]. Credential cloud sync is critical for satisfying end-user expectations of recoverability, but cloud sync means that the practical security of synced credentials reduces to the security of the cloud account, including cloud provider security [170, 123, 279, 389].

Perhaps most importantly, while the credentials themselves are encrypted in E2EE systems, credential vault metadata is generally unencrypted [211]. The precise categories of metadata stored in plaintext vary by service, but E2EE cloud services have repeatedly been shown to be vulnerable to metadata and other injection attacks exploiting file deduplication as well as other storage features [211, 153]. LastPass's 2022 compromise was particularly concerning because of the large amount of unencrypted metadata [267, 454], prompting LastPass to begin encrypting more metadata such as URLs [120].

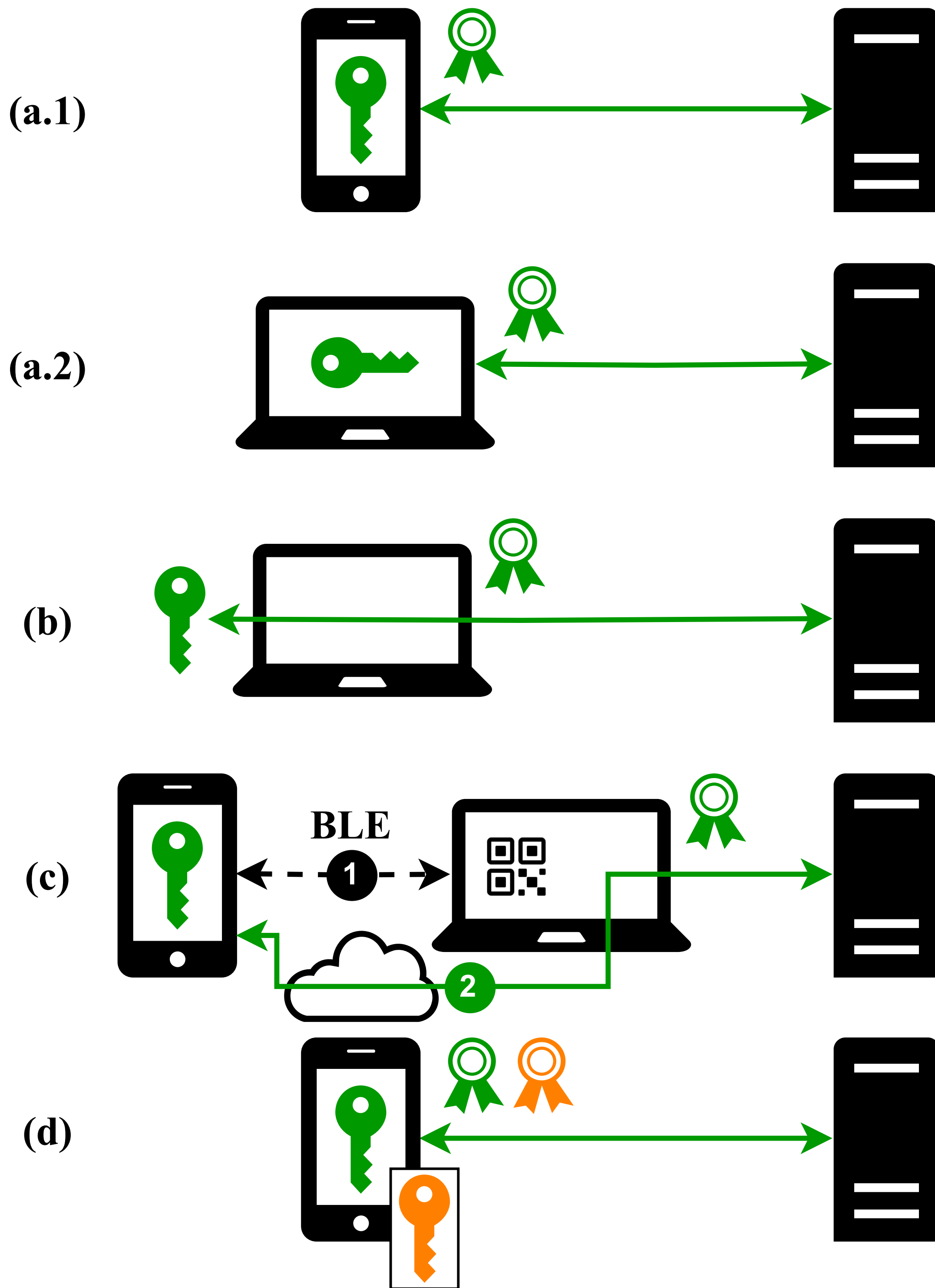

Figure 6.2: Passkey authentication flows: (a.1) standard authentication using a passkey on a mobile phone and (a.2) computer; (b) authentication with a roaming authenticator (NFC, BLE, or USB); (c) hybrid cross-device authentication initiated via a QR code; (d) authentication with an additional credential using the *device-bound public key extension*.

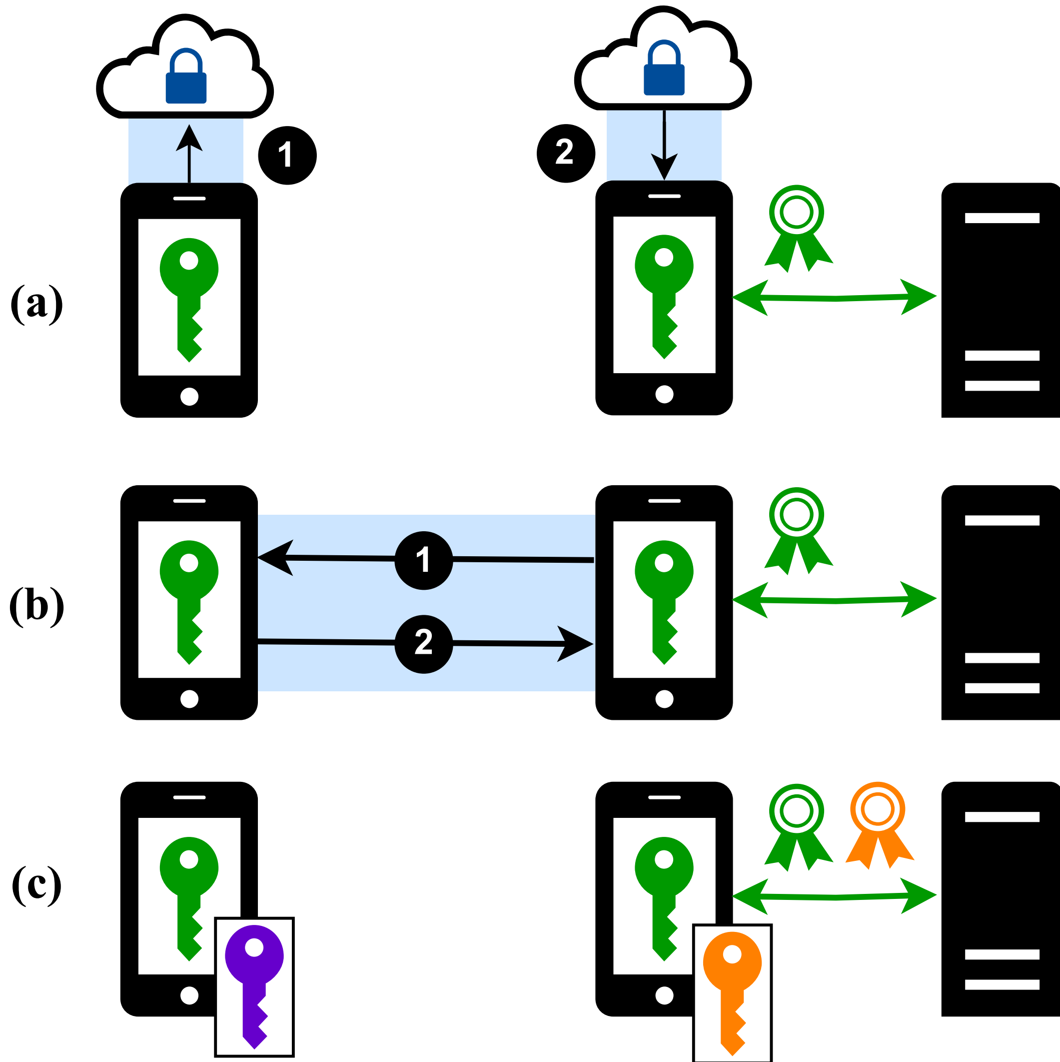

Figure 6.3: Passkey synchronization: (a) synchronization via the cloud (either E2EE between enclaves or using a third-party app); (b) importing into a new device using CXP; (c) in either approach, device-bound keys are not synchronized.

6.4 Passkeys

Major industry vendors (namely, Apple and Google) refer to the FIDO2 passwordless authentication mechanism as *passkeys*. [170] Google first introduced passkey support for Google Accounts in May 2023 [119] and Apple followed suit in June [443], though both will continue to support passwords as an authentication mechanism for the foreseeable future. On Apple devices, credentials such as passkeys are part of the keychain information that can be backed up to iCloud using Apple's in-house E2EE iCloud Keychain [73]. Passkey syncing to iCloud Keychain occurs by default but users can disable this synchronization for individual devices. On Android devices, credentials such as passkeys are managed by the optionally E2EE Google Password Manager which provides E2EE backups that can be restored on new devices.

All major operating systems and web browsers now support passwordless authentication [170], including cross-device authentication (e.g. using a phone with a computer that has no local authenticator) and cross-device sync using a platform-agnostic password manager such as 1Password [12].

6.4.1 Authentication

During registration and sign-in flows a *relying party* (e.g., a webservice) interacts with the device, the *client*, on which the user wishes to authenticate. This is mediated via the WebAuthn protocol [455] such that the relying party generally does not have to be concerned with the authenticator implementation and characteristics. The client-to-authenticator protocol (CTAP) [163] then enables communication between the client and the authenticator. We illustrate common authentication flows in Figure 6.2. The simplest authentication variant uses a *platform authenticator* where the client device itself has direct access to an internal authenticator, e.g., an iOS device storing passkeys in its Secure Enclave (Figure 6.2a). Alternatively, the client might use a discrete hardware device, a *roaming authenticator*, connected via NFC, BLE, or USB (Figure 6.2b).

6.4.2 Cross-Device Authentication

The latest FIDO2 standard draft allows cross-device *authentication* (CDA), where a passkey on one device can be used to authenticate to a service from a different device, including a device from a different vendor ecosystem (Figure 6.2c). We consider a typical first-time CDA flow where the user tries to login to a website on a laptop using an existing passkey on their smartphone [164, §11.5.1]. When receiving the WebAuthn request, the

[1]While it is possible to sync credentials directly between an on-device TEE and a server TEE with remote attestation, this is generally only feasible when both TEEs are controlled by the same provider (namely, when the device vendor is the same as the cloud service vendor, such as Apple or Google).

client, i.e. the web browser, displays a QR code which is scanned with the smartphone. The smartphone then shares BLE announcements to prove proximity and contribute to the shared secret state between itself and the client. Both now establish a connection via a tunnel service through which the authentication request is forwarded from the client to the authenticator. The client and authenticator may remember this link and subsequent visits can initiate a connection based on the state and without scanning a QR code [164, §11.5.2].

6.4.3 Backup and Recovery

Most platforms support first-party backup of passkeys which is typically limited to devices of the same ecosystem (see §6.3.2). Alternatively, the user might manage their passkeys using third-party CMs. In the first-party scenario, passkeys are typically only processed in secure hardware whereas it is common for third-party applications to perform operations on the decrypted credential in the regular application space (Figure 6.3). The latter setup can expose them while in-use to malware running on the device even if they are encrypted-at-rest with help of a secure element.

The authenticator decides during creation of a passkey whether it allow backups and/or synchronization between devices. For each credential, the authenticator attests to the relying party whether the credential is *backup-eligible*. Credentials which are backup eligible are commonly referred to as a *multi-device key*. Authenticators also report whether the credential is currently backed up, which relying parties can make use of to improve the user experience. For instance, a website might inform the user that their credential is also available on other devices; or it might warn them that their credentials are currently not backed up and that they should set up additional recovery mechanisms.

Since the backup eligibility is chosen by the authenticator, the relying party often cannot determine whether the same device is used across two authentication attempts. Where device binding is important, the relying party can use the *device-bound public key extension* [455, §10.2.2] to ask the authenticator to create an additional public key with the credential (see Figure 6.2d). The additional public key is never synced or backed up (see Figure 6.3c) allowing the relying party to identify new devices.

6.4.4 Exchange Between Known Devices

In addition to backups, which allow for future recovery with a fresh device, users may wish to share a passkey between two known devices. While this can be achieved with existing backup-and-recover flows, an interactive protocol between the devices allows for a simpler and more secure exchange. The Credential Exchange Protocol (CXP) [166] describes how an importing device can send an export request to another device which then allows it to

wrap the credential using a Hybrid Public Key Encryption (HPKE) scheme [33] and sends it over to the new device. CXP's flexible authorization design allows to make exchange depend on approval from other parties. For instance, a company can use it to ensure that its employees can forward certain passkeys only to other company-owned devices.

6.4.5 Passkey Usability

Academic studies of end-user perceptions of passwordless authentication revealed inaccurate mental models (e.g. believing the fingerprint or other biometric data is sent to the web server [265]) and challenges with inconsistencies in user interface design [343, 345], though passwordless authentication was generally considered simple to set up and use [460]. Users repeatedly express concerns over account loss [460, 283], though cloud sync recovery was not an option at the time the studies were conducted (and so users were presented with a design in which total device loss meant total account loss). After reviewing usability studies of passwordless authentication and current FIDO2 deployments, we identify three residual concerns hampering passkey adoption: sharing, revocation and availability.

6.4.5.1 Credential Sharing

Credential sharing is a common practice across numerous scenarios, including home and work environments [460, 438, 280, 250, 392, 459]. The latest industry deployments have made passkey sharing simpler with inter-ecosystem syncing, but there are still several cases where existing syncing mechanisms are inadequate [88]. One of the contexts in which credential sharing is most prevalent, the workplace, is particularly challenging given the recent trend towards remote work such that users may not be in physical proximity to each other, making passwords far simpler to share over common written communication channels. While there are a handful of vendor-specific solutions (for instance, passkeys can now be AirDropped via iCloud Keychain in iOS [74]) these are niche and not relevant for many use cases.

6.4.5.2 Credential Revocation

The FIDO2 standard does not adequately address credential revocation at a global scale. Currently, it assumes web services will implement the ability for a user to revoke access (e.g., remove the public key from the server) for specific authenticators, which a user will need to do for each individual website for which they have registered an authenticator [460, 283].

6.4.5.3 Credential Use Cases

That passkeys can be bound to the trusted hardware of specific devices (sometimes by default) is a major advantage from a security standpoint but can pose a significant

disadvantage for widespread usability and availability, particularly for at-risk demographics. Shared devices (including public computers [324]) are common, with users sharing devices for financial, cultural, and personal reasons [97, 245, 259, 24, 105, 350], including temporary sharing (e.g., showing a friend or relative a photo slideshow [245, 304]). Conversely, this also raises privacy concerns over inadvertent account sharing depending on passkey access duration before requiring reauthentication in a particular implementation.

6.4.6 Passkey Deployment and Availability

Here, we quantify the deployment of passwordless authentication in practice since the FIDO2 standard was published in 2019 [165]. In 2019, Ulqinaku et al. [420] found that 23 of the Alexa top 100 websites support the Universal 2nd Factor (U2F) hardware token protocol, but did not find any sites supporting passwordless authentication at the time. More recently, in 2023 Kuchhal et al. [261] surveyed the prevalence of the Web Authentication (WebAuthn) API used to provide public-key authentication (and deployed as part of various MFA schemes and passwordless authentication), finding that while 85 of the 585 domains in the Tranco Top 1K [356] that supported account creation also supported the WebAuthn protocol, the vast majority supported MFA, not passwordless authentication.

To measure passkey deployment in 2024, we manually inspect each site in the top 300 of the Alexa Top 1M dataset of most widely visited domains [242]. For sites that offer multiple account versions (e.g. a free version and a paid version), we follow the methodology of Gavazzi et al. [177] and select what we expect to be the most common account type. We observed that several sites do not offer passkey generation at the time of account creation, forcing a user to register with a standard username/password combination if not using single sign-on (See Appendix 2.3.1.1), but do allow a user to create a passkey as an additional authentication mechanism when logging in a second time. All experiments were conducted by logging in from Chrome 124.0.6367.203 on macOS Sonoma 14.2.1 in November 2024 from Cambridge, U.K.

Of the top 300 domains, we successfully audited $n = 206$; the remaining sites either did not offer account creation, required service-specific information to set up the account such as a phone number with a particular country code or banking credentials, or did not load. Since 44 of these sites are Google country-specific domains (e.g. *google.ch*) which use the same Google account we eliminate these from our dataset leaving us with $n = 162$ domains. Of these 162 sites, we find 17 sites (10.5%) offer direct passkey support for user authentication. We observe that a further 70 sites (43.2%) do not directly support passkeys but offer single-sign on with a provider which *does* support passkeys (in the vast majority of cases, Google). If we include sites which offer indirect passkey support through SSO, 87 of 162, or 53.7%, of sites in the top 300 directly or indirectly offer passkey

support, a significant increase from a 2021 study which found no support for passwordless authentication among 235 popular sites [177].

6.4.7 Passkeys and E2EE systems

Passkeys are not a feasible authentication or recovery mechanism for E2EE systems given that passkey schemes authenticate against a third-party and do not provide key material locally (making it all too easy for a user to lose access to the passkey). In fact, it is the other way around: passkeys rely on strong E2EE systems to become usable through cross-device synchronization and backup mechanisms. However, we believe that passkeys hold important lessons for developing and evaluating usable E2EE systems. Researchers, developers, and designers should take a close look at the already identified usability problems that emerge when keys are not directly accessible, but can only be handled within the constraints of existing standards. Unlike a password or recovery code, cryptographic keys are inherently intangible and in case of secure hardware not even accessible to users. Therefore, establishing correct and helpful mental models is essential. In particular, exchanging keys between two known, online devices is much simpler than restoring a backup to a new device at a later time without access to the original device. As a general lesson, passkeys demonstrate that stronger security at reasonable usability cost is often easier to achieve within one homogeneous ecosystem. Requiring inter-ecosystem interoperability can lead to lower overall security where secure hardware requires proprietary access and first-party authorization.

6.5 Summary

Overall, syncable passkeys provide lower security guarantees than device-bound passkeys. Not all passkeys are created equal: the FIDO Alliance uses the term “passkey” to refer to any passwordless FIDO credential [169, 167], but there are important security distinctions [329]. While device-bound credentials by definition never leave the hardware enclave in which they are created, E2EE syncable credentials potentially travel over multiple cloud providers to other client devices. The ability to sync credentials at all is essential for usability [167] and made possible by E2EE, but E2EE is not a panacea: passkeys are vulnerable to all the same issues impacting E2EE cloud storage and credential managers in the past, including metadata, brute-force attacks, and other client-side malware [211, 153, 454]. This is especially true if the user credential vault is protected by a weak password (widespread weak master password use was one of the reasons the 2022 LastPass server compromise was concerning [267, 260]). Only iCloud Keychain syncing provides a similar security level as device-bound credentials as discussed in Section 6.3.2.

Even so, syncable passkeys provide significant security improvements over even a randomly-generated password. Credential manager GUIs are generally designed such that the passkey's cryptographic key pair cannot even be viewed by the user even if they wanted to provide enhanced phishing-resistance [11]. Passkeys further prevent users from using weak passwords or reusing passwords since the credential is auto-generated for them.

Finally, biometric authentication schemes may provide improved security from a purely technical standpoint, there are important legal precedents in the U.S. governing when law enforcement can compel a user to unlock a device: while passwords and other forms of knowledge-based authentication are generally protected by the Fourth and Fifth Amendments (i.e. law enforcement cannot require an individual to provide their device passcode to guard against unreasonable search and seizure), multiple district-level courts in the U.S. have ruled that law enforcement can forcibly compel fingerprint authentication [233]. A user for whom preventing law enforcement access is the most important aspect of their personal threat model may intentionally avoid biometric authentication even though it provides greater protection against a generic external adversary.

Chapter 7

Conclusion and Future Work

The aim of this dissertation is to explore ways of building more secure E2EE *systems*, which goes well beyond the core E2EE *protocol*. Chapter 3 systematized the ways E2EE messaging platforms can be designed or modified to be made less secure in a case study of interoperable messaging. In particular, interoperability without robust identity management, moderation, and human-focused interface design to make platforms pleasant to use is a nonstarter. Giving users a choice between platforms without giving them a platform they would want to spend time on is no choice at all. The challenges in this space, though, are all the more reason for researchers to devote effort and attention to it, to ensure that users continue to have secure channels of communication in an interoperable world.

Chapters 4 and 5 build on our interoperable messaging case study through presenting two specific security design choices relevant for E2EE services: key storage and key recovery. Both of these design paradigms are deeply interconnected as using hardware-backed key storage can make account recovery more challenging if the relevant keys are stored only in a single device's hardware. There are countless individual use cases and unique scenarios prompting system designers to weigh the intersection of security, privacy, and usability differently when E2EE is deployed. Therefore, rather than attempt specific recommendations, we instead focus on presenting quantitative information around design choices relevant to E2EE that have historically been overlooked in prior work. We present the first comprehensive, large-scale survey of trusted hardware usage and performance in Android devices, providing developers with concrete performance data to encourage adoption and ultimately to enable them to make an informed decision. Similarly, in Chapter 5 we argue that E2EE providers should offer users a greater choice of recovery options to reflect the diversity of users' situations and perceived account value now that E2EE is increasingly targeted at the general public.

Above all, the PGP-era approach that users will handle retaining their own decryption keys (including recovery keys) remains the predominant strategy in use today. Although the vast majority of contemporary services hide the precise cryptographic keys from the

user, these services nonetheless require the user to retain and secure whichever abstraction is used to derive the keys (e.g., password, physical device, etc.). While this degree of user responsibility may be a fair assumption with the risk assessment of E2EE messaging backups, it does not pass muster with general cloud storage or authentication credential backups. Failure to provide users with a greater diversity of recovery options (which users have expressed a desire for [212]) may make non-technical users reluctant to opt in to advanced security schemes. Moreover, for most users the prospect of account lockout is itself a security threat, where loss of access to important files, credentials, or other data will have a significant impact on real-world security and well-being.

The net result will be that E2EE services (for data of high importance to users, like cloud storage) are largely confined to a smaller subset of dedicated users, ultimately limiting the security benefits for the general public. Cloud service providers may be reluctant to offer certain recovery options out of concern over increasing the attack surface. But E2EE cloud storage with more usable recovery options is still more secure than non-E2EE cloud storage, where a user is *guaranteed* that at least one third-party, the service provider itself, can access their account data. Perhaps most importantly from the service provider's perspective, a more user-friendly recovery process will go a long way towards mitigating the number of irate customers who become dissatisfied and switch to a competitor, particularly in competitive markets like mobile handsets or credential managers.

7.1 Future Work

Security analyses of interoperable systems: The initial interoperable messaging schemes proposed by WhatsApp and Facebook Messenger [313] are largely stopgap proposals to satisfy the European Union's timeline. There is substantial work left to be done in developing more privacy-preserving cryptographic protocols for the specific needs of cross-provider sharing as described in Chapter 3. If interoperable schemes are eventually deployed (meaning smaller services have both agreed to interoperate with Meta and dedicated the resources to modifying their own system), there will be ample opportunity for systems researchers to search for vulnerabilities and other flaws in real-world deployments.

Need for public performance evaluations of StrongBox: Android StrongBox's execution time may be entirely reasonable in cases with very small payloads: for instance, an app may use StrongBox to encrypt a different cryptographic key. Equally, developers may evaluate overhead cost differently depending on whether it is a one-time operation (e.g., initial login) or a repeated process. Developers need quantitative information in order to make case-by-case decisions, a gap which our work fills. Most importantly, Android's documentation

arguably understates the depth of the performance drawbacks of SEs, making it more challenging for developers to make an informed decision. Updated, empirical performance measurements based on contemporary device measurements should be publicly available and easily accessible to developers in place of the ambiguous language currently used in the documentation, which may cause developers to opt out of using StrongBox as a precautionary measure.

User perceptions and understanding of E2EE recovery. Prior work in E2EE recovery has shown some users request recovery schemes which are common in everyday web authentication but incompatible with E2EE (e.g., manual reset, security questions, or email/SMS recovery) [212]. Users have also demonstrated poor comprehension of the distinction between account recovery (e.g., the ability to log in) and recovering storage content (e.g., email history, calendar) [212]. To date there has been just one academic study of E2EE recovery [212] and more user interface design research is needed to confirm existing findings and to better understand how to improve user understanding around E2EE recovery. In particular, future work should consider investigating the relative importance users assign to different E2EE services (i.e. how users perceive loss of ephemeral messaging history compared with credential manager loss and loss of cloud storage) and the impact on types of recovery schemes a provider may want to offer.

Longitudinal recovery studies: Account recovery mechanisms are, by definition, more likely to be needed as more time has passed. In 2015, Bonneau et al. [101] found a linear relationship between the time passed since account creation and the proportion of authentication reset requests. Despite this, most usability studies conduct a cross-sectional examination of recovery schemes at a single point in time. A recent longitudinal study from Lassak et al. [266] found that for common non-E2EE recovery schemes (email, SMS, recovery questions, and social authentication) the relative convenience of recovery is consistent over time, but similar studies are critical for E2EE services, where the most common E2EE recovery scheme is to require users to maintain a non-human-memorable recovery code. A survey of E2EE email recovery support threads [212] found strong evidence suggesting users are likely to misplace their recovery code over time, motivating further work quantifying this possibility and deriving best practices. Evaluating an optimal balance between mandatory and opt-in periodic backup code confirmation (across a spectrum of backup codes from short PINs to arbitrary pseudorandom recovery codes) is another critical area of future research.

Comparative analysis of different social authentication schemes: Despite being deployed as one of Apple's E2EE storage recovery choices, trusted contact recovery has received comparatively little attention from the academic research community. The need for a greater focus here is particularly critical in light of recent developments in generative

AI, leaving end-users highly vulnerable to impersonation scams and account compromise, a reality made all the more dangerous given that deployed authentication schemes are increasingly centralized within a single account.

There are several variations in deployed trusted contact authentication schemes in E2EE services. Some schemes rely on a single trusted contact, some allow users to specify multiple individual trusted contacts, and another (PreVeil) has deployed group recovery where a threshold number of contacts must participate. Providers also sometimes also enforce time delays after trusted contact authentication has taken place. To date there have not been any studies investigating user *preferences* within these variations to understand what the general public would find most usable.

Improve security of E2EE credential syncing: A broad category of future work is to improve the security properties provided by E2EE credential managers. This includes metadata-hiding protocols for E2EE credential stores and cloud storage more generally, further analysis of backend third-party CM architectures, and improving interoperability of cross-vendor secure enclaves to enhance end-device security.

Bibliography

[1] Android Ready SE. https://developers.google.com/android/security/android-ready-se, 2021. Last accessed August 25th 2024.

[2] Soot. https://soot-oss.github.io/soot/, 2022.

[3] Have I Been Pwned. https://haveibeenpwned.com/, 2024. Last accessed 30th May 2024.

[4] Mozilla Monitor. https://monitor.mozilla.org/, 2024. Last accessed 24th Nov 2024.

[5] Cryptography in Mobile Apps. https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04g-testing-cryptography, 2024.

[6] AWS Device Farm. https://aws.amazon.com/device-farm/, 2025. Last accessed April 24th, 2025.

[7] Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design, October 2023.

[8] 1Password. Viewing passkeys. https://1password.community/discussion/142844/viewing-passkeys, 2023. October 2023.

[9] 1Password. Best practices for securing your 1Password Business account. https://support.1password.com/business-security-practices/, 2024. Last accessed November 29th, 2024.

[10] 1Password. 1Password Security Design. https://1passwordstatic.com/files/security/1password-white-paper.pdf, 2024. June 25, 2024.

[11] 1Password. How can I see my actual passkey? https://1password.community/discussion/148371/how-can-i-see-my-actual-passkey, 2024. September 20th, 2024.

[12] 1Password. Now available: Save and sign in with passkeys using 1Password in the browser and on iOS. https://blog.1password.com/save-use-passkeys-web-ios/, September 2023.

[13] Jacob Abbott and Sameer Patil. How mandatory second factor affects the authentication user experience. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[14] Jacob Abbott, Daniel Calarco, and L Jean Camp. Factors influencing password reuse: A case study. TPRC, 2018.

[15] Hadi Abdullah, Kevin Warren, Vincent Bindschaedler, Nicolas Papernot, and Patrick Traynor. Sok: The faults in our asrs: An overview of attacks against automatic speech recognition and speaker identification systems. In *2021 IEEE Symposium on Security and Privacy (S&P)*, pages 730–747. IEEE, 2021.

[16] Hal Abelson, Ross Anderson, Steven M Bellovin, Josh Benaloh, Matt Blaze, Whitfeld Diffie, John Gilmore, Peter G Neumann, Ronald L Rivest, Jeffrey I Schiller, et al. The risks of key recovery, key escrow, and trusted third-party encryption. 1997.

[17] Hal Abelson, Ross Anderson, Steven M Bellovin, Josh Benaloh, Matt Blaze, Jon Callas, Whitfield Diffie, Susan Landau, Peter G Neumann, Ronald L Rivest, et al. Bugs in our pockets: The risks of client-side scanning. *arXiv preprint arXiv:2110.07450*, 2021.

[18] Harold Abelson, Ross Anderson, Steven M Bellovin, Josh Benaloh, Matt Blaze, Whitfield "Whit" Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G Neumann, et al. Keys under doormats. *Communications of the ACM*, 58(10):24–26, 2015.

[19] Ruba Abu-Salma, Elissa M Redmiles, Blase Ur, and Miranda Wei. Exploring User Mental Models of End-to-End Encrypted Communication Tools. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, 2018.

[20] AccessNow. PUBLIC SECURITY ALERT: New Facebook attack – watch out for phishy messages that say you're a "Trusted Contact". https://www.accessnow.org/public-security-alert-new-facebook-attack/, 2017.

[21] Adam Gabbatt. How the domestic terror plot to kidnap Michigan's governor unravelled. https://www.theguardian.com/us-news/2020/oct/08/michigan-governor-gretchen-whitmer-kidnap-plot, 2020. October 9, 2020.

[22] Alaadin Addas, Amirali Salehi-Abari, and Julie Thorpe. Geographical security questions for fallback authentication. In *2019 17th international conference on privacy, security and trust (PST)*, pages 1–6. IEEE, 2019.

[23] Age Verification Providers Association. US State age verification laws for adult content. `https://avpassociation.com/4271-2/`, 2024. Last accessed November 20th, 2024.

[24] Syed Ishtiaque Ahmed, Md Romael Haque, Irtaza Haider, Jay Chen, and Nicola Dell. “everyone has some personal stuff” designing to support digital privacy with shared mobile phone use in bangladesh. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[25] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *USENIX security symposium*, volume 13, pages 257–272, 2013.

[26] Fatma Al Maqbali and Chris J Mitchell. Email-based password recovery-risking or rescuing users? In *2018 International Carnahan Conference on Security Technology (ICCST)*, pages 1–5. IEEE, 2018.

[27] Yusuf Albayram and Mohammad Maifi Hasan Khan. Evaluating smartphone-based dynamic security questions for fallback authentication: a field study. *Human-Centric Computing and Information Sciences*, 6:1–35, 2016.

[28] Martin R Albrecht, Sofía Celi, Benjamin Dowling, and Daniel Jones. Practically-exploitable cryptographic vulnerabilities in matrix. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 164–181. IEEE, 2023.

[29] Alec Muffett. A Civil Society Glossary and Primer for End-to-End Encryption Policy in 2022. `https://alecmuffett.com/alecm/e2e-primer/e2e-primer-web.html`, 2022.

[30] Reem AlHusain and Ali Alkhalifah. Evaluating fallback authentication research: A systematic literature review. *Computers & Security*, 111:102487, 2021.

[31] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[32] Noura Alomar, Mansour Alsaleh, and Abdulrahman Alarifi. Social authentication applications, attacks, defense strategies and future research directions: a systematic review. *IEEE Communications Surveys & Tutorials*, 19(2):1080–1111, 2017.

[33] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the hpke standard. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–116. Springer, 2021.

[34] Sabrina Amft, Sandra Höltervennhoff, Nicolas Huaman, Yasemin Acar, and Sascha Fahl. "would you give the same priority to the bank and a game? i do Not!" exploring credential management strategies and obstacles during password manager setup. In *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, pages 171–190, 2023.

[35] Sabrina Amft, Sandra Höltervennhoff, Nicolas Huaman, Alexander Krause, Lucy Simko, Yasemin Acar, and Sascha Fahl. "we've disabled mfa for you": An evaluation of the security and usability of multi-factor authentication recovery deployments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3138–3152, 2023.

[36] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2020.

[37] Ross Anderson. Chat control or child protection? *arXiv preprint arXiv:2210.08958*, 2022.

[38] Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. A large-scale empirical analysis of browser fingerprints properties for web authentication. *ACM Transactions on the Web (TWEB)*, 16(1):1–62, 2021.

[39] Android. AndroidKeyStoreBCWorkaroundProvider.java. `https://android.googlesource.com/platform/frameworks/base/+/marshmallow-mr1-release/keystore/java/android/security/keystore/AndroidKeyStoreBCWorkaroundProvider.java`, .

[40] Android. Remediation for unsafe encryption mode usage. `https://support.google.com/faqs/answer/10046138`, .

[41] Android. Verify hardware-backed key pairs with Key Attestation. `https://developer.android.com/privacy-and-security/security-key-attestation`, .

[42] Android. KeyGenParameterSpec. `https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec`, .

[43] Android. KeyProtection. `https://developer.android.com/reference/android/security/keystore/KeyProtection`, .

[44] Android. MasterKey. https://developer.android.com/reference/androidx/security/crypto/MasterKey, .

[45] Android. Security guidelines. https://developer.android.com/privacy-and-security/security-tips, .

[46] Android. setAttestationChallenge. https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setAttestationChallenge(byte[]), .

[47] Android. setRandomizedEncryptionRequired. https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setRandomizedEncryptionRequired(boolean), .

[48] Android. setUserAuthenticationRequired. https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setUserAuthenticationRequired(boolean), .

[49] Android. SharedPreferences. https://developer.android.com/reference/android/content/SharedPreferences, .

[50] Android. Hardware security module. https://developer.android.com/privacy-and-security/keystore#HardwareSecurityModule, .

[51] Android. Trusty TEE. https://source.android.com/docs/security/features/trusty, .

[52] Android. androidx.security.crypto. https://developer.android.com/reference/androidx/security/crypto/package-summary, .

[53] Android. App Security Improvement Program. https://developer.android.com/privacy-and-security/googleplay-asi, .

[54] Android. Cipher. https://developer.android.com/reference/javax/crypto/Cipher, .

[55] Android. KeyStore. https://developer.android.com/reference/java/security/KeyStore, .

[56] Android. KEY_ALGORITHM_3DES. https://developer.android.com/reference/android/security/keystore/KeyProperties#KEY_ALGORITHM_3DES, 2024.

[57] Android. KeyProperties. https://developer.android.com/reference/android/security/keystore/KeyProperties, 2024.

[58] Android. Android Keystore system. https://developer.android.com/privacy-and-security/keystore, 2024.

[59] Android Developers. Cryptography. https://developer.android.com/privacy-and-security/cryptography, 2024.

[60] Android Developers. Jelly Bean. https://developer.android.com/about/versions/jelly-bean, 2024.

[61] Android Developers. setBlockModes. https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setBlockModes(java.lang.String[]), 2024.

[62] Android Developers. setBlockModes. https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setEncryptionPaddings(java.lang.String[]), 2024.

[63] Android Developers. Conscrypt. https://source.android.com/docs/core/ota/modular-system/conscrypt, 2025.

[64] Apktool. Apktool. https://apktool.org/, 2024.

[65] Apple. Apple advances user security with powerful new data protections. https://www.apple.com/newsroom/2022/12/apple-advances-user-security-with-powerful-new-data-protections/, 2022.

[66] Apple. If you can't remember the password for your encrypted backup. https://support.apple.com/en-us/HT213037, 2023.

[67] Apple. Help a friend or family member as their account recovery contact, 2023. https://support.apple.com/en-us/102608.

[68] Apple. Set up a recovery key for your Apple ID. https://support.apple.com/en-us/109345, 2023.

[69] Apple. iCloud Keychain security overview. https://support.apple.com/en-gb/guide/security/sec1c89c6f3b/1/web/1, 2024. Last accessed 28th November 2024.

[70] Apple. Secure iCloud Keychain recovery. https://support.apple.com/en-gb/guide/security/secdeb202947/1/web/1, 2024. Last accessed 19th June 2024.

[71] Apple. Secure keychain syncing. https://support.apple.com/en-gb/guide/security/sec0a319b35f/1/web/1, 2024. Last accessed 28th November 2024.

[72] Apple. How to use account recovery when you can't reset your Apple ID password. https://support.apple.com/en-us/118574, 2024. Last accessed 30th May 2024.

[73] Apple. Passkeys Overview. https://developer.apple.com/passkeys/, 2024. Last accessed 28th May 2024.

[74] Apple. Share passkeys and passwords securely with AirDrop on iPhone. https://support.apple.com/en-gb/guide/iphone/iph0dd1796bb/ios, 2024. Last accessed 29th November 2024.

[75] Apptentive. apptentive/android/sdk/encryption/resolvers/KeyResolver23.java. https://github.com/apptentive/apptentive-android/blob/91aebf3fa758edddd40924f06aecdf1be4f12683/apptentive/src/main/java/com/apptentive/android/sdk/encryption/resolvers/KeyResolver23.java#L70, 2019.

[76] René Arnold, Anna Schneider, and Jonathan Lennartz. Interoperability of interpersonal communications services–a consumer perspective. *Telecommunications Policy*, 44(3):101927, 2020.

[77] American Medical Association. HIPAA Security Rule & Risk Analysis. https://www.ama-assn.org/practice-management/hipaa/hipaa-security-rule-risk-analysis, 2025.

[78] AWS-SDK-Android. amazonaws/internal/keyvaluestore/KeyProvider23.java. https://github.com/aws-amplify/aws-sdk-android/blob/8fd69db5e22d13973ddebf6521f5663ae2275c4c/aws-android-sdk-core/src/main/java/com/amazonaws/internal/keyvaluestore/KeyProvider23.java#L91, 2019.

[79] Azeem Azhar. How to Regulate Facebook (with Nick Clegg). https://hbr.org/podcast/2021/06/how-to-regulate-facebook-with-nick-clegg, 2021.

[80] Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, and Jin Song Dong. Authscan: Automatic extraction of web authentication protocols from implementations. 2013.

[81] Daniel V Bailey, Philipp Markert, and Adam J Aviv. "i have no idea what they're trying to accomplish:" enthusiastic and casual signal users' understanding of signal PINs. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 417–436, 2021.

[82] David G Balash, Xiaoyuan Wu, Miles Grant, Irwin Reyes, and Adam J Aviv. Security and privacy perceptions of Third-Party application access for google accounts. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3397–3414, 2022.

[83] Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) protocol. *Internet Engineering Task Force, Internet-Draft draft-ietf-mls-architecture/*, 2020.

[84] Andrey Belenko and Dmitry Sklyarov. "secure password managers" and "military-grade encryption" on smartphones: Oh, really? *Blackhat Europe*, page 56, 2012.

[85] Bennett Cyphers and Cory Doctorow. Privacy Without Monopoly: Data Protection and Interoperability. `https://www.eff.org/wp/interoperability-and-privacy`, February 2021.

[86] BEREC. BEREC report on interoperability of Number-Independent Interpersonal Communication Services (NI-ICS), December 2022.

[87] Antonio Bianchi, Yanick Fratantonio, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, Simon Pak Ho Chung, and Wenke Lee. Broken Fingers: On the Usage of the Fingerprint API in Android. In *NDSS*, 2018.

[88] Kemal Bicakci and Yusuf Uzunay. Is fido2 passwordless authentication a hype or for real?: A position paper. In *2022 15th International Conference on Information Security and Cryptography (ISCTURKEY)*, pages 68–73. IEEE, 2022.

[89] Bitkey. Losing your keys without losing your coins. `https://bitkey.build/losing-your-keys-without-losing-your-coins/`, 2022.

[90] Bitwarden. Bitwarden Security Whitepaper. `https://bitwarden.com/help/bitwarden-security-white-paper/`, 2024. Last accessed 29th November 2024.

[91] Bitwarden. Account Recovery. `https://bitwarden.com/help/account-recovery/`, 2024. Last accessed November 29th, 2024.

[92] Sam Blackshear, Konstantinos Chalkias, Panagiotis Chatzigiannis, Riyaz Faizullabhoy, Irakliy Khaburzaniya, Eleftherios Kokoris Kogias, Joshua Lind, David Wong, and Tim Zakian. Reactive key-loss protection in blockchains. In *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*, pages 431–450. Springer, 2021.

[93] Jenny Blessing and Ross Anderson. One Protocol to Rule Them All? On Securing Interoperable Messaging. In *Cambridge International Workshop on Security Protocols*, pages 174–192. Springer, 2023.

[94] Jenny Blessing, Daniel Hugenroth, Ross J Anderson, and Alastair R Beresford. Sok: Web authentication in the age of end-to-end encryption. *arXiv preprint arXiv:2406.18226*, 2024.

[95] Jenny Blessing, Michael A Specter, and Daniel J Weitzner. Cryptography in the wild: An empirical analysis of vulnerabilities in cryptographic libraries. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 605–620, 2024.

[96] Jenny Blessing, Ross J Anderson, and Alastair R Beresford. Keydroid: A large-scale analysis of secure key storage in android apps. *arXiv preprint arXiv:2507.07927*, 2025.

[97] Joshua Blumenstock and Nathan Eagle. Mobile divides: gender, socioeconomic status, and mobile phone use in rwanda. In *Proceedings of the 4th ACM/IEEE international conference on information and communication technologies and development*, pages 1–10, 2010.

[98] Joseph Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552. IEEE, 2012.

[99] Joseph Bonneau and Sören Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *WEIS*, 2010.

[100] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE, 2012.

[101] Joseph Bonneau, Elie Bursztein, Ilan Caron, Rob Jackson, and Mike Williamson. Secrets, lies, and account recovery: Lessons from the use of personal knowledge questions at google. In *Proceedings of the 24th international conference on world wide web*, pages 141–150, 2015.

[102] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84, 2004.

[103] Davide Bove. A Large-Scale Study on the Prevalence and Usage of TEE-based Features on Android. *arXiv preprint arXiv:2311.10511*, 2023.

[104] John Brainard, Ari Juels, Ronald L Rivest, Michael Szydlo, and Moti Yung. Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 168–178, 2006.

[105] Jenna Burrell. Evaluating shared access: social equality and the circulation of mobile phones in rural uganda. *Journal of computer-mediated communication*, 15(2): 230–250, 2010.

[106] Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean-Michel Picod, and Marina Zhang. Generic attacks against cryptographic hardware through long-range deep learning. *arXiv preprint arXiv:2306.07249*, 2023.

[107] Andre Büttner, Andreas Thue Pedersen, Stephan Wiefling, Nils Gruschka, and Luigi Lo Iacono. Is it really you who forgot the password? when account recovery meets risk-based authentication. In *International Conference on Ubiquitous Security*, pages 401–419. Springer, 2023.

[108] Michele Campobasso and Luca Allodi. Impersonation-as-a-service: Characterizing the emerging criminal infrastructure for user impersonation at scale. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1665–1680, 2020.

[109] Casey Newton. Three ways the European Union might ruin WhatsApp. `https://www.platformer.news/p/three-ways-the-european-union-might?s=r`, 2022.

[110] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-Assisted TEE Systems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1416–1432. IEEE, 2020.

[111] Charles Bethea). The Terrifying A.I. Scam That Uses Your Loved One's Voice. `https://www.newyorker.com/science/annals-of-artificial-intelligence/the-terrifying-ai-scam-that-uses-your-loved-ones-voice`, 2023. March 2024.

[112] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1639–1656, 2019.

[113] Melissa Chase, Hannah Davis, Esha Ghosh, and Kim Laine. Acsesor: A new framework for auditable custodial secret storage and recovery. *Cryptology ePrint Archive*, 2022.

[114] Rahul Chatterjee, Joseph Bonneau, Ari Juels, and Thomas Ristenpart. Cracking-resistant password vaults using natural language encoders. In *2015 IEEE Symposium on Security and Privacy*, pages 481–498. IEEE, 2015.

[115] Panagiotis Chatzigiannis, Konstantinos Chalkias, Aniket Kate, Easwar Vivek Mangipudi, Mohsen Minaei, and Mainack Mondal. Sok: Web3 recovery mechanisms. *Cryptology ePrint Archive*, 2023.

[116] Efstratios Chatzoglou, Vyron Kampourakis, Zisis Tsiatsikas, Georgios Karopoulos, and Georgios Kambourakis. Keep your memory dump shut: Unveiling data leaks in password managers. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 61–75. Springer, 2024.

[117] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157. IEEE, 2019.

[118] Eugene Cho, Jinyoung Kim, and S Shyam Sundar. Will you log into tinder using your facebook account? adoption of single sign-on for privacy-sensitive apps. In *Extended abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2020.

[119] Christiaan Brand and Sriram Karra. The beginning of the end of the password. May 2023.

[120] Christofer Hoff. LastPass is Encrypting URLs. Here's What's Happening. `https://blog.lastpass.com/posts/lastpass-is-encrypting-urls-heres-whats-happening`, 2024. May 22, 2024.

[121] Stéphane Ciolino, Simon Parkin, and Paul Dunphy. Of two minds about Two-Factor: Understanding everyday FIDOU2F usability through device comparison and experience sampling. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 339–356, 2019.

[122] Jessica Colnago, Summer Devlin, Maggie Oates, Chelse Swoopes, Lujo Bauer, Lorrie Cranor, and Nicolas Christin. "it's not actually that horrible" exploring adoption of

two-factor authentication at a university. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2018.

[123] Graeme Connell, Vivian Fang, Rolfe Schmidt, Emma Dauterman, and Raluca Ada Popa. Secret key recovery in a global-scale end-to-end encryption system. *Cryptology ePrint Archive*, 2024.

[124] Tim Cooijmans, Joeri de Ruiter, and Erik Poll. Analysis of secure key storage solutions on android. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 11–20, 2014.

[125] Cory Doctorow. Interoperable Facebook: How to Ditch Facebook Without Losing Friends. `https://www.eff.org/files/2022/09/12/interoperablefacebook-2022_0.pdf`, September 2022.

[126] Darren Loucaides. The Kremlin Has Entered the Chat. `https://www.wired.com/story/the-kremlin-has-entered-the-chat/`, 2023.

[127] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *Proceedings of the Symposium on Network and Distributed System Security*, volume 14, pages 23–26, 2014.

[128] Sanchari Das, Andrew Dingman, and L Jean Camp. Why johnny doesn't use two factor a two-phase usability study of the fido u2f security key. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pages 160–179. Springer, 2018.

[129] Sanchari Das, Bingxing Wang, Andrew Kim, and L Jean Camp. Mfa is a necessary chore!: Exploring user mental models of multi-factor authentication technologies. In *HICSS*, pages 1–10, 2020.

[130] Dashlane. Admin-assisted recovery for members of professional plans. `https://support.dashlane.com/hc/en-us/articles/115005111905-Admin-assisted-recovery-for-members-of-professional-plans`, 2024. Last accessed November 29th, 2024.

[131] Dashlane. Dashlane's Security Principles & Architecture. `https://www.dashlane.com/download/whitepaper-en.pdf`, 2024. June 10, 2024.

[132] Emma Dauterman, Henry Corrigan-Gibbs, and David Mazières. SafetyPin: Encrypted backups with Human-Memorable secrets. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1121–1138, 2020.

[133] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Greg Norcie. A comparative usability study of two-factor authentication. *arXiv preprint arXiv:1309.5344*, 2013.

[134] Yana Dimova, Tom Van Goethem, and Wouter Joosen. Everybody's looking for ssomething: A large-scale evaluation on the privacy of oauth authentication on the web. *Proceedings on Privacy Enhancing Technologies*, 2023.

[135] Dominik Schurmann. Why 2FA was useless with LastPass. `https://www.heylogin.com/en/post/lastpass-2fa-useless`, 2023. March 2, 2023.

[136] Antonin Durey, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-redemption: Studying browser fingerprinting adoption for the sake of web security. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings 18*, pages 237–257. Springer, 2021.

[137] Jonathan Dutson, Danny Allen, Dennis Eggett, and Kent Seamons. Don't punish all of us: measuring user attitudes about two-factor authentication. In *2019 IEEE European Symposium on Security and Privacy workshops (EuroS&PW)*, pages 119–128. IEEE, 2019.

[138] Ed Felten. How Yahoo could have protected Palin's email. `https://freedom-to-tinker.com/2008/09/21/how-yahoo-could-have-protected-palins-email/`, 2024. September 2008.

[139] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84, 2013.

[140] Element. The EU Digital Markets Act is here. `https://element.io/blog/the-eu-digital-markets-act-is-here/`, March 2024.

[141] Ellen Jennings-Trace. The European Commission wants a backdoor for end-to-end encryptions for law enforcement. `https://www.techradar.com/pro/security/the-european-commission-wants-a-backdoor-for-end-to-end-encryptions-for-law-e` 2025. April 2, 2025.

[142] Enpass. Access Recovery in Enpass Business. `https://support.enpass.io/business/console/hub/access_recovery_in_enpass_business.htm`, 2024. Last accessed November 29th, 2024.

[143] Enpass. Enpass Security Whitepaper. https://support.enpass.io/docs/security-whitepaper-enpass/index.html, 2024. Last accessed November 30th, 2024.

[144] Eric Rescorla. End-to-End Encryption and Messaging Interoperability. https://educatedguesswork.org/posts/messaging-e2e/, 2022.

[145] Eric Rescorla. Discovery Mechanisms for Messaging and Calling Interoperability. https://educatedguesswork.org/posts/messaging-discovery/, 2022.

[146] Eric Rescorla. Architectural Options for Messaging Interoperability. https://educatedguesswork.org/posts/dma-interop/, 2023.

[147] Yimika Erinle, Yathin Kethepalli, Yebo Feng, and Jiahua Xu. Sok: Design, vulnerabilities and defense of cryptocurrency wallets. *arXiv preprint arXiv:2307.12874*, 2023.

[148] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. End-to-end encrypted messaging protocols: An overview. In *Internet Science: Third International Conference, INSCI 2016, Florence, Italy, September 12-14, 2016, Proceedings 3*, pages 244–254. Springer, 2016.

[149] European Commission. Digital Markets Act. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32022R1925, 2023.

[150] European Commission. DMA workshop - The DMA and interoperability between messaging services. https://competition-policy.ec.europa.eu/dma/dma-workshops/interoperability-workshop_en, February 2023.

[151] European Parliament. Combating child sexual abuse online. https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/738224/EPRS_BRI(2022)738224_EN.pdf, May 2022.

[152] evmbrahmin.eth. Passkeys in Apple's iCloud Keychain. https://evmbrahmin.com/blog/iCloud-keychain-passkeys-guide.html, 2024. Last accessed November 30th, 2024.

[153] Andrés Fábrega, Armin Namavari, Rachit Agarwal, Ben Nassi, and Thomas Ristenpart. Exploiting leakage in password managers via injection attacks. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4337–4354, 2024.

[154] Facebook. Messenger Secret Conversations: Technical Whitepaper. https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf, 2016.

[155] Facebook. Choose friends to help you log in if you ever get locked out of your account. 2023.

[156] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 50–61, 2012.

[157] Florian M Farke, Lennart Lorenz, Theodor Schnitzler, Philipp Markert, and Markus Dürmuth. "You still use the password after all"–exploring FIDO2 security keys in a small company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 19–35, 2020.

[158] Shehroze Farooqi, Fareed Zaffar, Nektarios Leontiadis, and Zubair Shafiq. Measuring and mitigating oauth access token abuse by collusion networks. In *Proceedings of the 2017 Internet Measurement Conference*, pages 355–368, 2017.

[159] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638, 2011.

[160] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, pages 1–14, 2012.

[161] Adrienne Porter Felt, Alex Ainslie, Robert W Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. Improving ssl warnings: Comprehension and adherence. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 2893–2902, 2015.

[162] Daniel Fett, Ralf Küsters, and Guido Schmitz. Spresso: A secure, privacy-respecting single sign-on system for the web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1358–1369, 2015.

[163] FIDO Alliance. Client to Authenticator Protocol (CTAP). `https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html`, 2021. Last accessed 25th June 2024.

[164] FIDO Alliance. Client to Authenticator Protocol (CTAP). `http://fidoalliance.org/specs/fido-v2.2-rd-20230321/`

fido-client-to-authenticator-protocol-v2.2-rd-20230321.html, 2021. Last accessed 28th November 2024.

[165] FIDO Alliance. FIDO2. https://fidoalliance.org/fido2/, 2024. Last accessed 28th May 2024.

[166] FIDO Alliance. Credential Exchange Protocol. https://fidoalliance.org/specs/cx/cxp-v1.0-wd-20241003.html, 2024. Last accessed 28th November 2024.

[167] FIDO Alliance. White Paper: Synced Passkey Deployment: Emerging Practices for Consumer Use Cases. https://fidoalliance.org/white-paper-synced-passkey-deployment-emerging-practices-for-consumer-use-cases/, 2024. January 2024.

[168] FIDO Alliance. FIDO Universal 2nd Factor (U2F) Overview. https://fidoalliance.org/specs/u2f-specs-master/fido-u2f-overview.html, 2024. Last accessed 28th May 2024.

[169] FIDO Alliance. Passkeys. https://fidoalliance.org/passkeys/#faq, 2024. Last accessed November 30th, 2024.

[170] FIDO Alliance. How FIDO Addresses a Full Range of Use Cases. https://fidoalliance.org/wp-content/uploads/2022/03/How-FIDO-Addresses-a-Full-Range-of-Use-Cases-March24.pdf, March 2022.

[171] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666, 2007.

[172] Dinei Florêncio, Cormac Herley, and Baris Coskun. Do strong web passwords accomplish anything? *HotSec*, 7(6):159, 2007.

[173] Dinei Florêncio, Cormac Herley, and Paul C Van Oorschot. Password portfolios and the Finite-Effort user: Sustainably managing large numbers of accounts. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 575–590, 2014.

[174] Riccardo Focardi, Francesco Palmarini, Graham Steel, M Squarcina, Mauro Tempesta, et al. Mind your keys? a security evaluation of java keystores. In *Proceedings of the Network and Distributed System Security Symposium*, pages 1–15. The Internet Society, 2018.

[175] Schubert Foo, Siu Cheung Hui, Peng Chor Leong, and Shigong Liu. An integrated help desk support for customer services over the world wide web—a case study. *Computers in Industry*, 41(2):129–145, 2000.

[176] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. Who are you? a statistical approach to measuring user authenticity. In *Proceedings of the Symposium on Network and Distributed System Security*, volume 16, pages 21–24, 2016.

[177] Anthony Gavazzi, Ryan Williams, Engin Kirda, Long Lu, Andre King, Andy Davis, and Tim Leek. A study of Multi-Factor and Risk-Based authentication availability. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2043–2060, 2023.

[178] Eva Gerlitz, Maximilian Häring, Charlotte Theresa Mädler, Matthew Smith, and Christian Tiefenau. Adventures in recovery land: Testing the account recovery of popular websites when the second factor is lost. In *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, pages 227–243, 2023.

[179] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. O single Sign-Off, where art thou? an empirical analysis of single Sign-On account hijacking and session management on the web. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1475–1492, 2018.

[180] Mohammad Ghasemisharif, Chris Kanich, and Jason Polakis. Towards automated auditing for account and session management flaws in single sign-on deployments. In *2022 IEEE Symposium on Security and Privacy (S&P)*, pages 1774–1790. IEEE, 2022.

[181] Conor Gilsenan, Fuzail Shakir, Noura Alomar, and Serge Egelman. Security and privacy failures in popular {2FA} apps. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2079–2096, 2023.

[182] Gina Kolata. The Key Vanishes: Scientist Outlines Unbreakable Code. `https://www.nytimes.com/2001/02/20/science/the-key-vanishes-scientist-outlines-unbreakable-code.html`, February 2001.

[183] GitHub. GitHub Search Results. `https://github.com/search?q=%22setIsStrongBoxBacked%28false%29%22+language%3AJava&type=code&l=Java&p=1`, 2025.

[184] Maximilian Golla and Markus Dürmuth. Analyzing 4 million real-world personal knowledge questions (short paper). In *Technology and Practice of Passwords: 9th International Conference, PASSWORDS 2015, Cambridge, UK, December 7–9, 2015, Proceedings 9*, pages 39–44. Springer, 2016.

[185] Maximilian Golla, Benedict Beuscher, and Markus Dürmuth. On the security of cracking-resistant password vaults. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1230–1241, 2016.

[186] Maximilian Golla, Miranda Wei, Juliette Hainline, Lydia Filipe, Markus Dürmuth, Elissa Redmiles, and Blase Ur. "what was that site doing with my facebook password?" designing password-reuse notifications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1549–1566, 2018.

[187] Maximilian Golla, Grant Ho, Marika Lohmus, Monica Pulluri, and Elissa M Redmiles. Driving 2FA adoption at scale: Optimizing Two-Factor authentication notification design patterns. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 109–126, 2021.

[188] Google. Security of passkeys in the Google password manager. `https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html`, 2022. Last accessed 28th May 2024.

[189] Google. How to recover your Google Account or Gmail. `https://support.google.com/accounts/answer/7682439?hl=en&sjid=6761530502411005413-EU`, 2024. Last accessed 30th May 2024.

[190] Google. Why your account recovery request is delayed. `https://support.google.com/accounts/answer/9412469?hl=en&sjid=6761530502411005413-EU`, 2024. Last accessed 30th May 2024.

[191] Google. Passkeys, Cross-Account Protection and new ways we're protecting your accounts. `https://blog.google/technology/safety-security/google-passkeys-update-april-2024/`, May 2024.

[192] Paul A Grassi, James L Fenton, Elaine M Newton, Ray Perlner, Andrew Regenscheid, William E Burr, Justin P Richer, Naomi Lefkovitz, Jamie M Danker, Yee-Yin Choong, et al. Digital identity guidelines: Authentication and lifecycle management [includes updates as of 03-02-2020]. 2020.

[193] Colin M Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L Toombs. The dark (patterns) side of ux design. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–14, 2018.

[194] Carla F Griggio, Midas Nouwens, and Clemens Nylandsted Klokmose. Caught in the Network: The Impact of WhatsApp's 2021 Privacy Policy Update on Users' Messaging App Ecosystems. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–23, 2022.

[195] Eric Grosse and Mayank Upadhyay. Authentication at scale. *IEEE Security & Privacy*, 11(1):15–22, 2012.

[196] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message Franking via Committing Authenticated Encryption. In *Annual International Cryptology Conference*, pages 66–97. Springer, 2017.

[197] Chengqian Guo, Jingqiang Lin, Quanwei Cai, Wei Wang, Fengjun Li, Qiongxiao Wang, Jiwu Jing, and Bin Zhao. Uppresso: Untraceable and unlinkable privacy-preserving single sign-on services. *arXiv preprint arXiv:2110.10396*, 2021.

[198] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[199] Britta Hale and Chelsea Komlo. On end-to-end encryption. *Cryptology ePrint Archive*, 2022.

[200] Alina Hang, Alexander De Luca, and Heinrich Hussmann. I know what you did last week! do you? dynamic security questions for fallback authentication on smartphones. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1383–1392, 2015.

[201] Alina Hang, Alexander De Luca, Matthew Smith, Michael Richter, and Heinrich Hussmann. Where have you been? using Location-Based security questions for fallback authentication. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 169–183, 2015.

[202] Alina Hang, Alexander De Luca, Emanuel Von Zezschwitz, Manuel Demmler, and Heinrich Hussmann. Locked your phone? buy a new one? from tales of fallback authentication on smartphones to actual concepts. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 295–305, 2015.

[203] Hao Chen, Zhiyi Zhang, Haozhi Xiong, Ananth Raghunathan). How Meta is improving password security and preserving privacy. `https://engineering.fb.com/2023/08/08/security/`

how-meta-is-improving-password-security-and-preserving-privacy/, 2023. August 2023.

[204] Hao Chen, Zhiyi Zhang, Haozhi Xiong, Ananth Raghunathan). Passwordless login with passkeys. https://developers.google.com/identity/passkeys, 2024. Last accessed 29th May 2024.

[205] Heather Chen and Kathleen Magramo. Finance worker pays out $25 million after video call with deepfake 'chief financial officer'. https://edition.cnn.com/2024/02/04/asia/deepfake-cfo-scam-hong-kong-intl-hnk/index.html, 2024. February 4, 2024.

[206] Sudhi Herle and Jason Wong. Announcing the Android Ready SE Alliance. https://security.googleblog.com/2021/03/announcing-android-ready-se-alliance.html, 2021.

[207] Cormac Herley and Paul Van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security & privacy*, 10(1):28–36, 2011.

[208] Cormac Herley, Paul C Van Oorschot, and Andrew S Patrick. Passwords: If we're so smart, why are we still using them? In *Financial Cryptography and Data Security: 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers 13*, pages 230–237. Springer, 2009.

[209] Brad Hill. Moving account recovery beyond email and the "secret" question. 2017.

[210] Matthew Hodgson. Update on Native Matrix interoperability with WhatsApp. https://matrix.org/blog/2024/09/whatsapp-dma/, September 16, 2024.

[211] Jonas Hofmann and Kien Tuong Truong. End-to-end encrypted cloud storage in the wild: A broken ecosystem. *Cryptology ePrint Archive*, 2024.

[212] Sandra Höltervennhoff, Noah Wöhler, Arne Möhle, Marten Oltrogge, Yasemin Acar, Oliver Wiese, and Sascha Fahl. A mixed-methods study on user experiences and challenges of recovery codes for an end-to-end encrypted service. In *In 33rd USENIX Security Symposium*, 2024.

[213] Daniel Hugenroth, Alberto Sonnino, Sam Cutler, and Alastair R Beresford. Sloth: Key stretching and deniable encryption using secure elements on smartphones. *Cryptology ePrint Archive*, 2023.

[214] Hugo Lowell. Exclusive: how the Atlantic's Jeffrey Goldberg got added to the White House Signal group chat. https://www.theguardian.com/us-news/2025/apr/06/signal-group-chat-leak-how-it-happened, 2025. April 6, 2025.

[215] Ian Brown. Private Messaging Interoperability in the EU Digital Markets Act. https://openforumeurope.org/wp-content/uploads/2022/11/Ian_Brown_Private_Messaging_Interoperability_In_The_EU_DMA.pdf, December 2022.

[216] Ian Levy and Crispin Robinson. Principles for a More Informed Exceptional Access Debate. https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate, 2018.

[217] IETF. More Instant Messaging Interoperability (MIMI). https://datatracker.ietf.org/wg/mimi/about/, 2025.

[218] Abdullah Imran, Habiba Farrukh, Muhammad Ibrahim, Z Berkay Celik, and Antonio Bianchi. SARA: Secure Android Remote Authorization. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1561–1578, 2022.

[219] Information Commissioner's Office. What is the eIDAS Regulation? https://ico.org.uk/for-organisations/guide-to-eidas/what-is-the-eidas-regulation/, 2024. Last accessed November 20th, 2024.

[220] Philip G Inglesant and M Angela Sasse. The true cost of unusable password policies: password use in the wild. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 383–392, 2010.

[221] Intel. What Is Virtualization Security? https://www.intel.com/content/www/us/en/business/enterprise-computers/resources/virtualization-security.html, 2025.

[222] Internet Society. White Paper: Considerations for Mandating Open Interfaces. https://www.internetsociety.org/wp-content/uploads/2020/12/ConsiderationsMandatingOpenInterfaces-03122020-EN.pdf, December 2020.

[223] Internet Society. DMA and interoperability of encrypted messaging. https://www.internetsociety.org/wp-content/uploads/2022/03/ISOC-EU-DMA-interoperability-encrypted-messaging-20220311.pdf, March 2022.

[224] ISE. Password Managers: Under the Hood of Secrets Management. https://www.ise.io/casestudies/password-manager-hacking/, 2019. February 19, 2019.

[225] Takanori Isobe and Ryoma Ito. Security analysis of end-to-end encryption for zoom meetings. *IEEE access*, 9:90677–90689, 2021.

[226] Ivan Krstić. Personal Data in the Cloud Is Under Siege. End-to-End Encryption Is Our Most Powerful Defense. https://www.lawfaremedia.org/article/personal-data-in-the-cloud-is-under-siege.-end-to-end-encryption-is-our-most-powerful-defense, 2023.

[227] Shubham Jain, Ana-Maria Crețu, and Yves-Alexandre de Montjoye. Adversarial detection avoidance attacks: Evaluating the robustness of perceptual hashing-based client-side scanning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2317–2334, 2022.

[228] James Humphreys. LATEST: Keeper Security's recovery feature. https://securityjournaluk.com/latest-keeper-securitys-recovery-feature/, 2023. April 28, 2023.

[229] Ashar Javed, David Bletgen, Florian Kohlar, Markus Dürmuth, and Jörg Schwenk. Secure fallback authentication and the trusted friend attack. In *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 22–28. IEEE, 2014.

[230] Joanna Stern and Nicole Nguyen. A Basic iPhone Feature Helps Criminals Steal Your Entire Digital Life. https://www.wsj.com/tech/personal-tech/apple-iphone-security-theft-passcode-data-privacya-basic-iphone-feature-helps-crim-steal-your-digital-life-cbf14b1a, February 2023.

[231] John Swanson. Raising the bar for software security: next steps for GitHub.com 2FA. https://github.blog/2022-12-14-raising-the-bar-for-software-security-next-steps-for-github-com-2fa/, December 2022.

[232] Trevor Johns. Using Cryptography to Store Credentials Safely. https://android-developers.googleblog.com/2013/02/using-cryptography-to-store-credentials.html, 2013.

[233] Jon Brodkin. Cops can force suspect to unlock phone with thumbprint, US court rules. https://arstechnica.com/tech-policy/2024/04/cops-can-force-suspect-to-unlock-phone-with-thumbprint-us-court-rules/, 2024. April 18th, 2024.

[234] Jonathan Rosenberg. A Taxonomy for More Messaging Interop (MIMI). https://datatracker.ietf.org/doc/draft-rosenberg-mimi-taxonomy/00/, October 2022.

[235] Jonathan Rosenberg and Cullen Jennings and Alissa Cooper and Jon Peterson. Simple Protocol for Inviting Numbers (SPIN). https://www.ietf.org/archive/id/draft-rosenberg-mimi-spin-00.html, October 2022.

[236] Joseph Menn. Apple dropped plan for encrypting backups after FBI complained. https://www.reuters.com/article/world/exclusive-apple-dropped-plan-for-encrypting-backups-after-fbi-complained/, 2020. January 22, 2020.

[237] Joseph Menn. U.K. orders Apple to let it spy on users' encrypted accounts. https://www.washingtonpost.com/technology/2025/02/07/apple-encryption-backdoor-uk/, 2025. February 7, 2025.

[238] Joseph Menn. U.K. demand for a back door to Apple data threatens Americans, lawmakers say. https://www.washingtonpost.com/technology/2025/02/13/apple-uk-security-back-door-adp/, 2025. February 13, 2025.

[239] Roger Piqueras Jover. Security analysis of sms as a second factor of authentication. *Communications of the ACM*, 63(12):46–52, 2020.

[240] Julia Angwin. Back into the Trenches of the Crypto Wars: A conversation with Meredith Whittaker. https://themarkup.org/hello-world/2023/01/07/back-into-the-trenches-of-the-crypto-wars, 2023.

[241] Mike Just and David Aspinall. Personal choice and challenge questions: a security and usability assessment. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–11, 2009.

[242] Kaggle. Alexa Top 1 Million Sites. https://www.kaggle.com/datasets/cheedcheed/top1m, 2024. Last accessed November 29th, 2024.

[243] Shirin Kalantari, Pieter Philippaerts, Yana Dimova, Danny Hughes, Wouter Joosen, and Bart De Decker. A user-centric approach to api delegations: Enforcing privacy policies on oauth delegations. In *European Symposium on Research in Computer Security*, pages 318–337. Springer, 2023.

[244] Seny Kamara, Mallory Knodel, Emma Llansó, Greg Nojeim, Lucy Qin, Dhanaraj Thakur, and Caitlin Vogus. Outside Looking In: Approaches to Content Moderation in End-to-End Encrypted Systems. *arXiv preprint arXiv:2202.04617*, 2022.

[245] Amy K Karlson, AJ Bernheim Brush, and Stuart Schechter. Can i borrow your phone? understanding concerns when sharing mobile phones. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1647–1650, 2009.

[246] Kashmir Hill. A Dad Took Photos of His Naked Toddler for the Doctor. Google Flagged Him as a Criminal. https://www.nytimes.com/2022/08/21/technology/google-surveillance-toddler-photo.html, 2022.

[247] Kashmir Hill. Her Child's Naked Dance Killed Her Google Account. New Appeals Path Restored It. https://www.nytimes.com/2022/12/30/technology/google-appeals-change.html, 2022.

[248] Kashmir Hill. How Your Child's Online Mistake Can Ruin Your Digital Life. https://www.nytimes.com/2023/11/27/technology/google-youtube-abuse-mistake.html, 2023.

[249] Andre Kassis and Urs Hengartner. Breaking security-critical voice authentication. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pages 951–968. IEEE, 2023.

[250] Joseph'Jofish' Kaye. Self-reported password sharing strategies. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 2619–2622, 2011.

[251] Keeper. Master Password Reset & Account Recovery. https://docs.keeper.io/en/user-guides/troubleshooting/reset-your-master-password, 2024. Last accessed November 29th, 2024.

[252] Keeper. Keeper Encryption and Security Model Details. https://docs.keeper.io/en/enterprise-guide/keeper-encryption-model, 2024. Last accessed November 30th, 2024.

[253] Rishabh Khandelwal, Asmit Nayak, Paul Chung, and Kassem Fawaz. Unpacking privacy labels: A measurement and developer perspective on google's data safety section. *arXiv preprint arXiv:2306.08111*, 2023.

[254] Kim Key. The Best Password Managers for 2024. https://uk.pcmag.com/password-managers/4296/the-best-password-managers, 2024. November 19, 2024.

[255] Kim Zetter. Palin E-Mail hacker Says It Was Easy. https://www.wired.com/2008/09/palin-e-mail-ha/, 2008.

[256] Jennifer King, Airi Lampinen, and Alex Smolen. Privacy: Is there an app for that? In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, pages 1–20, 2011.

[257] Dave Kleidermacher, Jesse Seed, Brandon Barbello, and Stephan Somogyi. Pixel 6: Setting a new standard for mobile security. `https://security.googleblog.com/2021/10/pixel-6-setting-new-standard-for-mobile.html`, 2021.

[258] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max Van Kleek, and Nigel Shadbolt. Are iphones really better for privacy? comparative study of ios and android apps. *arXiv preprint arXiv:2109.13722*, 2021.

[259] Leah Komen. "here you can use it": Understanding mobile phone sharing and the concerns it elicits in rural kenya. *for (e) dialogue*, 1(1):52–65, 2016.

[260] Krebs on Security. LastPass: 'Horse Gone Barn Bolted' is Strong Password. `https://krebsonsecurity.com/2023/09/lastpass-horse-gone-barn-bolted-is-strong-password/`, 2023. September 22, 2023.

[261] Dhruv Kuchhal, Muhammad Saad, Adam Oest, and Frank Li. Evaluating the security posture of real-world fido2 deployments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2381–2395, 2023.

[262] Anunay Kulshrestha and Jonathan R Mayer. Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In *USENIX Security Symposium*, pages 893–910, 2021.

[263] Johannes Kunke, Stephan Wiefling, Markus Ullmann, and Luigi Lo Iacono. Evaluation of account recovery strategies with fido2-based passwordless authentication. *arXiv preprint arXiv:2105.12477*, 2021.

[264] Daniele Lain, Kari Kostiainen, and Srdjan Čapkun. Phishing in organizations: Findings from a large-scale and long-term study. In *2022 IEEE Symposium on Security and Privacy (S&P)*, pages 842–859. IEEE, 2022.

[265] Leona Lassak, Annika Hildebrandt, Maximilian Golla, and Blase Ur. "it's stored, hopefully, on an encrypted server": Mitigating users' misconceptions about FIDO2 biometric WebAuthn. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 91–108, 2021.

[266] Leona Lassak, Philipp Markert, Maximilian Golla, Elizabeth Stobert, and Markus Dürmuth. A comparative long-term study of fallback authentication schemes. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2024.

[267] LastPass. 12-22-2022: Notice of Security Incident. https://blog.lastpass.com/posts/notice-of-recent-security-incident, 2022. December 22, 2022.

[268] LastPass. Emergency Access. https://www.lastpass.com/features/emergency-access, 2024. Last accessed 30th November 2024.

[269] LastPass. About the encryption process when a super admin resets a master password. https://support.lastpass.com/s/document-item?language=en_US&bundleId=lastpass&topicId=LastPass/reset-master-password-encryption.html&_LANG=enus, 2024. May 22, 2024.

[270] LastPass. Enable the "Permit super admins to reset master passwords" policy. https://support.lastpass.com/s/document-item?language=en_US&bundleId=lastpass&topicId=LastPass/t_lastpass_enable_super_admin_reset_masterpsw.html&_LANG=enus, 2024. July 11, 2024.

[271] LastPass. LastPass Technical Whitepaper. https://support.lastpass.com/s/document-item?language=en_US&bundleId=lastpass&topicId=LastPass/lastpass_technical_whitepaper.html&_LANG=enus, 2024. October 29, 2024.

[272] Kevin Lee and Arvind Narayanan. Security and privacy risks of number recycling at mobile carriers in the united states. In *2021 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–17. IEEE, 2021.

[273] Kevin Lee, Benjamin Kaiser, Jonathan Mayer, and Arvind Narayanan. An empirical study of wireless carrier authentication for SIM swaps. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 61–79, 2020.

[274] Julia Len, Esha Ghosh, Paul Grubbs, and Paul Rösler. Interoperability in end-to-end encrypted messaging. *Cryptology ePrint Archive*, 2023.

[275] Ian Levy and Crispin Robinson. Thoughts on child safety on commodity platforms. *arXiv preprint arXiv:2207.09506*, 2022.

[276] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 88: 67–95, 2017.

[277] Rui Li, Wenrui Diao, Zhou Li, Jianqi Du, and Shanqing Guo. Android custom permissions demystified: From privilege escalation to design shortcomings. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 70–86. IEEE, 2021.

[278] Yue Li, Haining Wang, and Kun Sun. Email as a master key: Analyzing account recovery in the wild. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1646–1654. IEEE, 2018.

[279] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The Emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 465–479, 2014.

[280] Junchao Lin, Jason I Hong, and Laura Dabbish. "it's our mutual responsibility to share" the evolution of account sharing in romantic couples. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–27, 2021.

[281] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. Phish in sheep's clothing: Exploring the authentication pitfalls of browser fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1651–1668, 2022.

[282] Lorenzo Franceschi-Biccherai. Stop Using 6-Digit iPhone Passcodes. `https://www.vice.com/en/article/how-to-make-a-secure-iphone-passcode-6-digits/`, 2018. April 16, 2018.

[283] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy (S&P)*, pages 268–285. IEEE, 2020.

[284] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. Libradar: Fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th international conference on software engineering companion*, pages 653–656, 2016.

[285] Carsten Maartmann-Moe, Steffen E Thorkildsen, and André Årnes. The persistence of memory: Forensic identification and extraction of cryptographic keys. *digital investigation*, 6:S132–S140, 2009.

[286] Christian Mainka, Vladislav Mladenov, Jörg Schwenk, and Tobias Wich. Sok: single sign-on security—an evaluation of openid connect. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 251–266. IEEE, 2017.

[287] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging. *Cryptology ePrint Archive*, 2023.

[288] Manuel Vonau. Signal and Threema want nothing to do with WhatsApp. `https://www.androidpolice.com/signal-threema-nothing-to-do-with-whatsapp-eu/`, February 2024.

[289] Marcela Melara. Why Making Johnny's Key Management Transparent is So Challenging. https://freedom-to-tinker.com/2016/03/31/why-making-johnnys-key-management-transparent-is-so-challenging/, 2016.

[290] Mark Risher. A simpler and safer future — without passwords. May 2021.

[291] Mark Zuckerberg. A Privacy-Focused Vision for Social Networking. https://www.nytimes.com/2019/03/06/technology/facebook-privacy-blog.html, 2019.

[292] Philipp Markert, Florian Farke, and Markus Dürmuth. View the email to get hacked: Attacking sms-based two-factor authentication. *Who Are You*, pages 1–6, 2019.

[293] Philipp Markert, Daniel V Bailey, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. This pin can be easily guessed: Analyzing the security of smartphone unlock pins. In *2020 IEEE Symposium on Security and Privacy (S&P)*, pages 286–303. IEEE, 2020.

[294] Philipp Markert, Daniel V Bailey, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. On the security of smartphone unlock pins. *ACM Transactions on Privacy and Security (TOPS)*, 24(4):1–36, 2021.

[295] Philipp Markert, Leona Lassak, Maximilian Golla, and Markus Dürmuth. Understanding users' interaction with login notifications. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2024.

[296] Karola Marky, Kirill Ragozin, George Chernyshov, Andrii Matviienko, Martin Schmitz, Max Mühlhäuser, Chloe Eghtebas, and Kai Kunze. "nah, it's just annoying!" a deep dive into user perceptions of two-factor authentication. *ACM transactions on computer-human interaction*, 29(5):1–32, 2022.

[297] Moxie Marlinspike. The ecosystem is moving. In *36th Chaos Communication Congress, Leipzig, Germany. https://www.youtube.com/watch?v=Nj3YFprqAr8&ab_channel=n99*, 2019.

[298] Matrix. Matrix Specification. https://spec.matrix.org/latest/, 2025.

[299] Matt Burgess. A New Era of Attacks on Encryption Is Starting to Heat Up. https://www.wired.com/story/a-new-era-of-attacks-on-encryption-is-starting-to-heat-up/, 2025.

[300] Matt Jones. How WhatsApp Reduced Spam While Launching End-to-End Encryption. https://www.youtube.com/watch?v=LBTOKlrhKXk&ab_channel=USENIXEnigmaConference, 2017.

[301] Matthew Hodgson. Interoperability without sacrificing privacy: Matrix and the DMA. https://matrix.org/blog/2022/03/25/interoperability-without-sacrificing-privacy-matrix-and-the-dma, 2022.

[302] Matthew Hodgson. How do you implement interoperability in a DMA world? https://matrix.org/blog/2022/03/29/how-do-you-implement-interoperability-in-a-dma-world, 2023.

[303] Matthew Hodgson. The DMA Stakeholder Workshop: Interoperability between messaging services. https://matrix.org/blog/2023/03/15/the-dma-stakeholder-workshop-interoperability-between-messaging-services, 2023.

[304] Tara Matthews, Kerwell Liao, Anna Turner, Marianne Berkovich, Robert Reeder, and Sunny Consolvo. "she'll just grab any device that's closer" a study of everyday device & account sharing in households. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5921–5932, 2016.

[305] Max Eddy). So You're Locked Out of Your Two-Factor Authentication App. Don't Panic. https://www.nytimes.com/wirecutter/guides/locked-out-two-factor-authentication-app-recovery/, 2024. April 2024.

[306] Jonathan Mayer. Content moderation for end-to-end encrypted messaging. *Princeton University*, 2019.

[307] René Mayrhofer, Jeffrey Vander Stoep, Chad Brubaker, and Nick Kralevich. The Android Platform Security Model. *ACM Transactions on Privacy and Security (TOPS)*, 24(3):1–35, 2021.

[308] mCodex. RNSensitiveInfoModule.java. https://github.com/mCodex/react-native-sensitive-info/blob/495dd7f08c077f5744e56803e45f54787df3dab3/android/src/main/java/dev/mcodex/RNSensitiveInfoModule.java#L313.

[309] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. CONIKS: Bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015.

[310] Meredith Whittaker. https://twitter.com/mer__edith/status/1582808091397005312, 2022.

[311] Meredith Whittaker. https://twitter.com/mer__edith/status/1629131348731478017, February 2023.

[312] Meta. The Labyrinth Encrypted Message Storage Protocol. https://engineering.fb.com/wp-content/uploads/2023/12/TheLabyrinthEncryptedMessageStorageProtocol_12-6-2023.pdf, 2023.

[313] Meta. Messaging Interoperability. https://developers.facebook.com/m/messaging-interoperability/, 2025.

[314] Meta. How WhatsApp is enabling end-to-end encrypted backups. https://engineering.fb.com/2021/09/10/security/whatsapp-e2ee-backups/, September 2021.

[315] Meta. An Update on How We're Building Safe and Secure Third-Party Chats for Users in Europe. https://about.fb.com/news/2024/09/an-update-on-how-were-building-safe-and-secure-third-party-chats-for-users-in-eur September 6, 2024.

[316] Microsoft. Require end-to-end encryption for sensitive Teams meetings. https://learn.microsoft.com/en-us/microsoftteams/end-to-end-encrypted-meetings, 2024. September 18, 2024.

[317] Grzergor Milka. Anatomy of account takeover. In *Enigma 2018 (Enigma 2018)*, 2018.

[318] Ariana Mirian, Joe DeBlasio, Stefan Savage, Geoffrey M Voelker, and Kurt Thomas. Hack for hire: Exploring the emerging market for account hijacking. In *The World Wide Web Conference*, pages 1279–1289, 2019.

[319] Srivathsan G Morkonda, Sonia Chiasson, and Paul C van Oorschot. Empirical analysis and privacy implications in oauth-based single sign-on systems. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, pages 195–208, 2021.

[320] Srivathsan G Morkonda, Sonia Chiasson, and Paul C van Oorschot. "sign in with... privacy": Timely disclosure of privacy differences among web sso login options. *arXiv preprint arXiv:2209.04490*, 2022.

[321] Moxie Marlinspike. Reflections: The ecosystem is moving. https://signal.org/blog/the-ecosystem-is-moving/, 2016.

[322] Moxie Marlinspike. There is no WhatsApp 'backdoor'. https://signal.org/blog/there-is-no-whatsapp-backdoor/, January 2017.

[323] Collin Mulliner, Ravishankar Borgaonkar, Patrick Stewin, and Jean-Pierre Seifert. Sms-based one-time passwords: Attacks and defense: (short paper). In *Detection of Intrusions and Malware, and Vulnerability Assessment: 10th International Conference, DIMVA 2013, Berlin, Germany, July 18-19, 2013. Proceedings 10*, pages 150–159. Springer, 2013.

[324] Collins W Munyendo, Yasemin Acar, and Adam J Aviv. "in eighty percent of the cases, i select the password for them": Security and privacy challenges, advice, and opportunities at cybercafes in kenya. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pages 570–587. IEEE, 2023.

[325] Collins W Munyendo, Peter Mayer, and Adam J Aviv. "i just stopped using one and started using the other": Motivations, techniques, and challenges when switching password managers. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3123–3137, 2023.

[326] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195. IEEE, 2005.

[327] Nate Cardozo. Making it Easier to Manage Business Conversations on WhatsApp. `https://about.fb.com/news/2020/10/privacy-matters-whatsapp-business-conversations/`, October 2020.

[328] Samsung Newsroom. Strengthening Hardware Security with Galaxy S20's Secure Processor. `https://news.samsung.com/global/strengthening-hardware-security-with-galaxy-s20s-secure-processor`, May 2020.

[329] Nicolas Bacca. Passkeys: The Good, The Bad, The Ugly. `https://www.youtube.com/watch?v=TEjNSr8jjUI&ab_channel=EthereumFoundation`, 2024. November 17, 2024.

[330] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A Survey of Published Attacks on Intel SGX. *arXiv preprint arXiv:2006.13598*, 2020.

[331] nina-signal. Removing SMS support from Signal Android (soon). `https://signal.org/blog/sms-removal-android/`, 2022.

[332] Alexandra Nisenoff, Maximilian Golla, Miranda Wei, Juliette Hainline, Hayley Szymanek, Annika Braun, Annika Hildebrandt, Blair Christensen, David Langenberg, and Blase Ur. A Two-Decade retrospective analysis of a university's vulnerability to attacks exploiting reused passwords. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5127–5144, 2023.

[333] Nora Trapp. Key to Simplicity: Squeezing the hassle out of encryption key recovery. https://juicebox.xyz/blog/key-to-simplicity-squeezing-the-hassle-out-of-encryption-key-recovery, 2024. April 9, 2024.

[334] NordPass. Account Recovery for business users. https://support.nordpass.com/hc/en-us/articles/360017323858-Account-Recovery-for-business-users, 2024. Last accessed November 29th, 2024.

[335] NordPass. Overview. https://support.nordpass.com/hc/en-us/sections/26090886272529-Overview, 2024. Last accessed November 30th, 2024.

[336] Midas Nouwens, Carla F Griggio, and Wendy E Mackay. " WhatsApp is for family; Messenger is for friends" Communication Places in App Ecosystems. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 727–735, 2017.

[337] Sean Oesch and Scott Ruoti. That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 2165–2182, 2020.

[338] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 1344–1361. IEEE, 2019.

[339] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[340] U.S. Department of Health and Human Services. Guidance to render unsecured protected health information unusable, unreadable, or indecipherable to unauthorized individuals. https://www.hhs.gov/hipaa/for-professionals/breach-notification/guidance/index.html, .

[341] U.S. Department of Health and Human Services. The security rule. https://www.hhs.gov/hipaa/for-professionals/security/index.html, .

[342] Marten Oltrogge, Yasemin Acar, Sergej Dechand, Matthew Smith, and Sascha Fahl. To pin or not to {Pin—Helping} app developers bullet proof their {TLS} connections. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 239–254, 2015.

[343] Wataru Oogami, Hidehito Gomi, Shuji Yamaguchi, Shota Yamanaka, and Tatsuru Higurashi. Observation study on usability challenges for fingerprint authentication using webauthn-enabled android smartphones. *Age*, 20:29, 2020.

[344] Chris Orsini, Alessandra Scafuro, and Tanner Verber. How to recover a cryptographic secret from the cloud. *Cryptology ePrint Archive*, 2023.

[345] Kentrell Owens, Olabode Anise, Amanda Krauss, and Blase Ur. User perceptions of the usability and security of smartphones as FIDO2 roaming authenticators. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, pages 57–76, 2021.

[346] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 417–434. IEEE, 2019.

[347] Simon Parkin, Samy Driss, Kat Krol, and M Angela Sasse. Assessing the user experience of password reset policies in a university. In *Technology and Practice of Passwords: 9th International Conference, PASSWORDS 2015, Cambridge, UK, December 7–9, 2015, Proceedings 9*, pages 21–38. Springer, 2016.

[348] Kenneth G Paterson, Matteo Scarlata, and Kien Tuong Truong. Three lessons from threema: Analysis of a secure messenger. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1289–1306, 2023.

[349] Patrick McGee. How Apple captured Gen Z in the US — and changed their social circles. `https://www.ft.com/content/8a2e8442-449e-4dbd-bd6d-2656b4503526`, February 2023.

[350] Rizu Paudel, Prakriti Dumaru, Ankit Shrestha, Huzeyfe Kocabas, and Mahdi Nasrullah Al-Ameen. A deep dive into user's preferences and behavior around mobile phone sharing. *Proceedings of the ACM on Human-Computer Interaction*, 7(CSCW1):1–22, 2023.

[351] Justin Petelka, Yixin Zou, and Florian Schaub. Put Your Warning Where Your Link Is: Improving and Evaluating Email Phishing Warnings. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–15, 2019.

[352] Thanasis Petsas, Giorgos Tsirantonakis, Elias Athanasopoulos, and Sotiris Ioannidis. Two-factor authentication: is the world ready? quantifying 2fa adoption. In *Proceedings of the eighth european workshop on system security*, pages 1–7, 2015.

[353] Riana Pfefferkorn. Content-oblivious trust and safety techniques: Results from a survey of online service providers. *Journal of Online Trust and Safety*, 1(2), 2022.

[354] Jamie L Pinchot and Karen L Paullet. What's in your profile? mapping facebook profile data to personal security questions. *Issues in Information Systems*, 13(1): 284–293, 2012.

[355] Sandro Pinto and Nuno Santos. Demystifying ARM TrustZone: A Comprehensive Survey. *ACM computing surveys (CSUR)*, 51(6):1–36, 2019.

[356] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156*, 2018.

[357] Jon Porter. Apple won't be forced to open up iMessage by EU. `https://www.theverge.com/2024/2/13/23990679/apple-imessage-european-union-digital-markets-act-core-platform-service`, February 13, 2024.

[358] Andrea Possemato and Yanick Fratantonio. Towards HTTPS everywhere on android: We are not there yet. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 343–360, 2020.

[359] PreVeil. PreVeil Security and Design: A Description of the PreVeil System Architecture. `https://www.preveil.com/wp-content/uploads/2019/10/Architectural_Whitepaper_FINAL.pdf`, 2019.

[360] Jonathan Prokos, Neil Fendley, Matthew Green, Roei Schuster, Eran Tromer, Tushar Jois, and Yinzhi Cao. Squint hard enough: Attacking perceptual hashing with adversarial machine learning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 211–228, 2023.

[361] Proton. Set account recovery methods in case you forget your Proton password. `https://proton.me/support/set-account-recovery-methods`, 2024.

[362] Nils Quermann, Marian Harbach, and Markus Dürmuth. The state of user authentication in the wild. *WAY*, 18, 2018.

[363] Ariel Rabkin. Personal knowledge questions for fallback authentication: Security questions in the era of facebook. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, pages 13–23, 2008.

[364] Rachel Hall. Apple to appeal against UK government data demand at secret high court hearing. https://www.theguardian.com/technology/2025/mar/12/apple-to-appeal-against-uk-government-data-demand-at-secret-high-court-hearing 2025. March 12, 2025.

[365] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX security symposium (USENIX security 19)*, pages 603–620, 2019.

[366] Robert W Reeder, Adrienne Porter Felt, Sunny Consolvo, Nathan Malkin, Christopher Thompson, and Serge Egelman. An experience sampling study of user reactions to browser warnings in the field. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–13, 2018.

[367] Joshua Reynolds, Trevor Smith, Ken Reese, Luke Dickinson, Scott Ruoti, and Kent Seamons. A tale of two studies: The best and worst of yubikey usability. In *2018 IEEE Symposium on Security and Privacy (S&P)*, pages 872–888. IEEE, 2018.

[368] Joshua Reynolds, Nikita Samarin, Joseph Barnes, Taylor Judd, Joshua Mason, Michael Bailey, and Serge Egelman. Empirical measurement of systemic 2FA usability. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 127–143, 2020.

[369] Hossein Rezaeighaleh and Cliff C Zou. New secure approach to backup cryptocurrency wallets. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[370] Phillip Rogaway. The moral character of cryptographic work. *Cryptology ePrint Archive*, 2015.

[371] Ryan Hurst and Gary Belvin. Security Through Transparency. https://security.googleblog.com/2017/01/security-through-transparency.html, 2017.

[372] Mohamed Sabt and Jacques Traoré. Breaking into the keystore: A practical forgery attack against android keystore. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II 21*, pages 531–548. Springer, 2016.

[373] Salesforce. KeyStoreWrapper.java. https://github.com/forcedotcom/SalesforceMobileSDK-Android/blob/1a11e225b20968cc88ed08cf3304ede28b5701af/libs/SalesforceSDK/src/com/salesforce/androidsdk/security/KeyStoreWrapper.java#L247, .

[374] Salesforce. Commit 4b074b7: Refactoring StrongBox code. `https://github.com/forcedotcom/SalesforceMobileSDK-Android/commit/4b074b7c744f44129486029a7df6481d0d7c3eb2`, .

[375] Samantha Cole. Google Is Begging Apple to Make Life Better for Green Bubbles. `https://www.vice.com/en/article/wxngb9/google-is-begging-apple-to-make-life-better-for-green-bubbles`, September 2022.

[376] Nithya Sambasivan, Garen Checkley, Amna Batool, Nova Ahmed, David Nemer, Laura Sanely Gaytán-Lugo, Tara Matthews, Sunny Consolvo, and Elizabeth Churchill. "privacy is not for me, it's for those rich women"': Performative privacy practices on mobile phones by women in south asia. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 127–142, 2018.

[377] Cristiana Santos, Arianna Rossi, Lorena Sanchez Chamorro, Kerstin Bongard-Blanchy, and Ruba Abu-Salma. Cookie banners, what's the purpose? analyzing cookie banner text through a legal lens. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, pages 187–194, 2021.

[378] Alessandra Scafuro. Break-glass encryption. In *Public-Key Cryptography–PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II 22*, pages 34–62. Springer, 2019.

[379] Stuart Schechter, Serge Egelman, and Robert W Reeder. It's not what you know, but who you know: a social approach to last-resort authentication. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1983–1992, 2009.

[380] Sarah Scheffler and Jonathan Mayer. Sok: Content moderation for end-to-end encryption. *arXiv preprint arXiv:2303.03979*, 2023.

[381] Svenja Schröder, Markus Huber, David Wind, and Christoph Rottermanner. When SIGNAL hits the fan: On the usability and security of state-of-the-art secure mobile messaging. In *European Workshop on Usable Security. IEEE*, pages 1–7, 2016.

[382] Fabian Schwarz, Khue Do, Gunnar Heide, Lucjan Hanzlik, and Christian Rossow. Feido: Recoverable fido2 tokens using electronic ids. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2581–2594, 2022.

[383] Sean Lawlor and Kevin Lewi. Deploying key transparency at WhatsApp. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/, 2023.

[384] Tobias Seitz, Manuel Hartmann, Jakob Pfab, and Samuel Souque. Do differences in password policies prevent password reuse? In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2056–2063, 2017.

[385] Asuman Senol, Alisha Ukani, Dylan Cutler, and Igor Bilogrevic. The double edged sword: Identifying authentication pages and their fingerprinting behavior. In *The Web Conference (WWW), 2024*, 2024.

[386] Alon Shakevsky, Eyal Ronen, and Avishai Wool. Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 251–268, 2022.

[387] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[388] Hossein Siadati, Toan Nguyen, Payas Gupta, Markus Jakobsson, and Nasir Memon. Mind your smses: Mitigating social engineering in second factor authentication. *Computers & Security*, 65:14–28, 2017.

[389] David Silver, Suman Jana, Dan Boneh, Eric Chen, and Collin Jackson. Password managers: Attacks and defenses. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 449–464, 2014.

[390] Silviu Stahie. Microsoft Teams Rolls Out End-to-End Encryption. https://www.bitdefender.com/en-gb/blog/hotforsecurity/microsoft-teams-rolls-out-end-to-end-encryption, 2021. October 25, 2021.

[391] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On breaking SAML: Be whoever you want to be. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 397–412, 2012.

[392] Yunpeng Song, Cori Faklaris, Zhongmin Cai, Jason I Hong, and Laura Dabbish. Normal and easy: Account sharing practices in the workplace. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–25, 2019.

[393] StartMail. I forgot my password - now what? https://support.startmail.com/hc/en-us/articles/360007388898--I-forgot-my-password-now-what, 2024. Last accessed 1 June 2024.

[394] Vlasta Stavova, Vashek Matyas, and Mike Just. Codes v. people: A comparative usability study of two password recovery mechanisms. In *Information Security Theory and Practice: 10th IFIP WG 11.2 International Conference, WISTP 2016, Heraklion, Crete, Greece, September 26–27, 2016, Proceedings 10*, pages 35–50. Springer, 2016.

[395] Elizabeth Stobert and Robert Biddle. The password life cycle. *ACM Transactions on Privacy and Security (TOPS)*, 21(3):1–32, 2018.

[396] Google Play Store. Provide information for Google Play's Data safety section. https://support.google.com/googleplay/android-developer/answer/10787469?hl=en, .

[397] Google Play Store. Signal Private Messenger. https://play.google.com/store/apps/datasafety?id=org.thoughtcrime.securesms&hl=en&gl=US, .

[398] Christian Stransky, Dominik Wermke, Johanna Schrader, Nicolas Huaman, Yasemin Acar, Anna Lena Fehlhaber, Miranda Wei, Blase Ur, and Sascha Fahl. On the limited impact of visualizing encryption: Perceptions of e2e messaging security. In *Seventeenth Symposium on Usable Privacy and Security*, pages 437–454, 2021.

[399] Sukhi Gulati-Gilbert and Michal Luria. Designing Interoperable, Encrypted Messaging with User Journeys. https://cdt.org/insights/designing-interoperable-encrypted-messaging-with-user-journeys/, August 2022.

[400] San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: an empirical analysis of oauth sso systems. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 378–390, 2012.

[401] San-Tsai Sun, Yazan Boshmaf, Kirstie Hawkey, and Konstantin Beznosov. A billion keys, but few locks: the crisis of web single sign-on. In *Proceedings of the 2010 new security paradigms workshop*, pages 61–72, 2010.

[402] San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. What makes users refuse web single sign-on? an empirical investigation of openid. In *Proceedings of the seventh symposium on usable privacy and security*, pages 1–20, 2011.

[403] San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 31(4):465–483, 2012.

[404] San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. Investigating users' perspectives of web single sign-on: Conceptual gaps and acceptance model. *ACM Transactions on Internet Technology (TOIT)*, 13(1):1–35, 2013.

[405] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX security symposium*, pages 399–416. Montreal, Canada, 2009.

[406] Yang Tang, Phillip Ames, Sravan Bhamidipati, Ashish Bijlani, Roxana Geambasu, and Nikhil Sarda. CleanOS: Limiting Mobile Data Exposure with Idle Eviction. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 77–91, 2012.

[407] The Alan Turing Institute. Trustchain. `https://github.com/alan-turing-institute/trustchain`, 2025.

[408] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, et al. Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1421–1434, 2017.

[409] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1556–1571, 2019.

[410] Threema. New Communication Protocol "Ibex" and Extended Protocol Suite. `https://threema.ch/en/blog/posts/ibex`, .

[411] Threema. Cryptography Whitepaper. `https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf`, .

[412] Threema. Anonymity – the ultimate privacy protection. `https://threema.ch/en/blog/posts/anonymity`, 2019.

[413] Tim Higgins. Why Apple's iMessage is Winning: Teens Dread the Green Text Bubble. `https://www.wsj.com/articles/why-apples-imessage-is-winning-teens-dread-the-green-text-bubble-11641618009`, January 2022.

[414] Travis Ralston and Matthew Hodgson. Matrix Message Transport. https://datatracker.ietf.org/doc/draft-ralston-mimi-matrix-transport/, 2022.

[415] Tresorit. End-to-end encryption without key rotation is a fatal shortcut. https://tresorit.com/blog/end-to-end-encryption-without-key-rotation-is-a-fatal-shortcut/, 2023. June 2, 2023.

[416] Zeynep Tufekci. In Response to Guardian's Irresponsible Reporting on WhatsApp: A Plea for Responsible and Contextualized Reporting on User Security, 2017. URL https://technosociology.org/?page_id=1687.

[417] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric Message Franking: Content Moderation for Metadata-Private End-to-End Encryption. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 222–250. Springer, 2019.

[418] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 413–430, 2019.

[419] Giannis Tzagarakis, Panagiotis Papadopoulos, Antonios A Chariton, Elias Athanasopoulos, and Evangelos P Markatos. Øpass: Zero-storage password management based on password reminders. In *Proceedings of the 11th European Workshop on Systems Security*, pages 1–6, 2018.

[420] Enis Ulqinaku, Hala Assal, AbdelRahman Abdou, Sonia Chiasson, and Srdjan Capkun. Is real-time phishing eliminated with FIDO? social engineering downgrade attacks against FIDO protocols. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3811–3828, 2021.

[421] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE, 2015.

[422] United Kingdom Department for Science, Innovation and Technology. Code of practice for app store operators and app developers. https://www.gov.uk/government/publications/code-of-practice-for-app-store-operators-and-app-developers, 2022.

[423] USA.gov. How to replace lost or stolen ID cards. https://www.usa.gov/replace-vital-documents, 2024. Last accessed November 30th, 2024.

[424] Warda Usman, Jackie Hu, McKynlee Wilson, and Daniel Zappala. Distrust of big tech and a desire for privacy: Understanding the motivations of people who have voluntarily adopted secure email. In *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, pages 473–490, 2023.

[425] Christine Utz, Martin Degeling, Sascha Fahl, Florian Schaub, and Thorsten Holz. (Un) Informed Consent: Studying GDPR Consent Notices in the Field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 973–990, 2019.

[426] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient Out-Of-Order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[427] Stephan Van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. Cacheout: Leaking data on intel cpus via cache evictions. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 339–354. IEEE, 2021.

[428] Adam Vartanian. Cryptography Changes in Android P. `https://android-developers.googleblog.com/2018/03/cryptography-changes-in-android-p.html`, March 8 2018.

[429] Diana A Vasile, Martin Kleppmann, Daniel R Thomas, and Alastair R Beresford. Ghost trace on the wire? using key evidence for informed decisions. In *Security Protocols XXVII: 27th International Workshop, Cambridge, UK, April 10–12, 2019, Revised Selected Papers 27*, pages 245–257. Springer, 2020.

[430] Elham Vaziripour, Justin Wu, Mark O'Neill, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala. Is that you, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 29–47, 2017.

[431] Elham Vaziripour, Justin Wu, Mark O'Neill, Daniel Metro, Josh Cockrell, Timothy Moffett, Jordan Whitehead, Nick Bonner, Kent E Seamons, and Daniel Zappala. Action Needed! Helping Users Find and Complete the Authentication Ceremony in Signal. In *SOUPS@ USENIX Security Symposium*, pages 47–62, 2018.

[432] Thijs Veugen, Robbert de Haan, Ronald Cramer, and Frank Muller. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE Transactions on Information Forensics and Security*, 10 (3):445–457, 2014.

[433] WABetaInfo. Apple is releasing password reminder for end-to-end encrypted backups. https://wabetainfo.com/whatsapp-is-releasing-password-reminder-feature-for-end-to-end-encrypted-backups/, 2023.

[434] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1242–1254, 2016.

[435] Ke Coby Wang and Michael K Reiter. How to end password reuse on the web. *arXiv preprint arXiv:1805.00566*, 2018.

[436] Qiwen Wang and Mikael Skoglund. Symmetric private information retrieval from mds coded distributed storage with non-colluding and colluding servers. *IEEE Transactions on Information Theory*, 65(8):5160–5175, 2019.

[437] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. Explicating SDKs: uncovering assumptions underlying secure authentication and authorization. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 399–314, 2013.

[438] Serena Wang, Cori Faklaris, Junchao Lin, Laura Dabbish, and Jason I Hong. 'it's problematic but i'm not concerned': University perspectives on account sharing. *Proceedings of the ACM on Human-Computer Interaction*, 6(CSCW1):1–27, 2022.

[439] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882, 2014.

[440] Rick Wash, Emilee Rader, Ruthie Berman, and Zac Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 175–188, 2016.

[441] Miranda Wei, Jaron Mink, Yael Eiger, Tadayoshi Kohno, Elissa M Redmiles, and Franziska Roesner. Sok (or solk?): On the quantitative study of sociodemographic factors and computer security behaviors. *arXiv preprint arXiv:2404.10187*, 2024.

[442] Mark Weiser. Program slicing. *IEEE Transactions on software engineering*, (4): 352–357, 1984.

[443] Wes Davis. Apple IDs now support passkeys — if you're on the iOS 17 or macOS Sonoma betas. June 2023.

[444] Maximilian Westers, Tobias Wich, Louis Jannett, Vladislav Mladenov, Christian Mainka, and Andreas Mayer. Sso-monitor: fully-automatic large-scale landscape, security, and privacy analyses of single sign-on in the wild. *arXiv preprint arXiv:2302.01024*, 2023.

[445] WhatsApp. About forwarding limits. https://faq.whatsapp.com/1053543185312573, .

[446] WhatsApp. What is traceability and why does WhatsApp oppose it? https://faq.whatsapp.com/1206094619954598, .

[447] WhatsApp. How WhatsApp Helps Fight Child Exploitation. https://faq.whatsapp.com/5704021823023684/?locale=en_US, .

[448] WhatsApp. Security of End-To-End Encrypted Backups. https://scontent-lhr8-1.xx.fbcdn.net/v/t39.8562-6/241394876_546674233234181_8907137889500301879_n.pdf?_nc_cat=108&ccb=1-7&_nc_sid=e280be&_nc_ohc=sei5qWjmFJgAX8WtvKt&_nc_ht=scontent-lhr8-1.xx&oh=00_AfA4yNP_d5B9LyiyOwX8N6MCDq6osRT3p4IflzrlEjEOzg&oe=65E2F3E6, 2021.

[449] WhatsApp. Can't remember password for encrypted backup . https://faq.whatsapp.com/639067727894647, 2023.

[450] Stephan Wiefling, Luigi Lo Iacono, and Markus Dürmuth. Is this really you? an empirical study on risk-based authentication applied in the wild. In *ICT Systems Security and Privacy Protection: 34th IFIP TC 11 International Conference, SEC 2019, Lisbon, Portugal, June 25-27, 2019, Proceedings 34*, pages 134–148. Springer, 2019.

[451] Stephan Wiefling, Markus Dürmuth, and Luigi Lo Iacono. More than just good passwords? a study on usability and security perceptions of risk-based authentication. In *Proceedings of the 36th Annual Computer Security Applications Conference*, pages 203–218, 2020.

[452] Stephan Wiefling, Paul René Jørgensen, Sigurd Thunem, and Luigi Lo Iacono. Pump up password security! evaluating and enhancing risk-based authentication on a real-world large-scale online service. *ACM Transactions on Privacy and Security*, 26 (1):1–36, 2022.

[453] Lukas Wiewiorra, Nico Steffen, Philipp Thoste, Niklas Fourberg, Serpil Taş, Peter Kroon, Christoph Busch, and Jan Krämer. Interoperability regulations for digital services: Impact on competition, innovation and digital sovereignty especially for platform and communication services. https://www.bundesnetzagentur.de/DE/Fachthemen/Digitalisierung/Technologien/Onlinekomm/Study_InteroperabilityregulationsDigiServices.pdf?__blob=publicationFile&v=1, August 2022.

[454] Wladimir Palant. What data does LastPass encrypt? https://palant.info/2022/12/24/what-data-does-lastpass-encrypt/, 2022. December 24, 2022.

[455] World Wide Web Consortium (W3C). Web Authentication: An API for accessing Public Key Credentials Level 3. https://www.w3.org/TR/webauthn-3/, 2023. Last accessed 28th May 2024.

[456] Worldcoin Foundation. Worldcoin – a new identity and financial network. https://whitepaper.worldcoin.org/, 2024. Last accessed 31st May 2024.

[457] Daoyuan Wu, Debin Gao, Robert H Deng, and Chang Rocky KC. When program analysis meets bytecode search: Targeted and efficient inter-procedural analysis of modern android apps in backdroid. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 543–554. IEEE, 2021.

[458] Shujiang Wu, Pengfei Sun, Yao Zhao, and Yinzhi Cao. Him of many faces: Characterizing billion-scale adversarial and benign browser fingerprints on commercial websites. In *Proceedings of the Symposium on Network and Distributed System Security*, 2023.

[459] Yuxi Wu, W Keith Edwards, and Sauvik Das. Sok: Social cybersecurity. In *2022 IEEE Symposium on Security and Privacy (S&P)*, pages 1863–1879. IEEE, 2022.

[460] Leon Würsching, Florentin Putz, Steffen Haesler, and Matthias Hollick. Fido2 the rescue? platform vs. roaming authentication on smartphones. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2023.

[461] Xiaowen Xin. Titan M makes Pixel 3 our most secure phone yet. https://blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/, 2018.

[462] Ronghai Yang, Wing Cheong Lau, Jiongyi Chen, and Kehuan Zhang. Vetting single Sign-OnSDK implementations via symbolic reasoning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1459–1474, 2018.

[463] Peng Zhao, Kaigui Bian, Tong Zhao, Xintong Song, Jung-Min Park, Xiaoming Li, Fan Ye, and Wei Yan. Understanding smartphone sensor and app data for enhancing the security of secret questions. *IEEE Transactions on Mobile Computing*, 16(2): 552–565, 2016.

[464] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326, 2012.

[465] Moshe Zviran and William J Haga. User authentication by cognitive passwords: an empirical assessment. In *Proceedings of the 5th Jerusalem Conference on Information Technology, 1990.'Next Decade in Information Technology'*, pages 137–144. IEEE, 1990.

Chapter 8

Secure Key Storage

8.1 Android App Manual Analysis

We select a subset of applications flagged as not using Android's trusted hardware API for further examination to verify our static analysis results and to better understand which key storage schemes are used instead. We scraped the top 200 most-downloaded apps in the Play Store as of April 1, 2024, and then selected the ten most highly-ranked apps which self-reported collecting sensitive data but had been flagged in our initial keyword search as not referencing the Android Keystore API. We decompiled each app using Apktool and manually searched for relevant keywords relating to widely used software-backed keystores provided as part of Android, such as Android's SharedPreferences API [49].

We verified that each of these ten apps were indeed not using Android's trusted hardware API anywhere and found that they instead generally made use of some combination of Android's SharedPreferences, Android's default software-backed keystore (i.e., `AndroidOpenSSL`), or a local SQLite database such as SQLCipher. SharedPreferences is a bit more concerning than other software-backed keystores as it offers very different security properties: some Android keystores (namely `SharedPreferences` and `KeyChain`) are intended as systemwide credential storage, where keys are accessible to *any* app on the device. While it is challenging to make any definitive statements on individual app use cases due to the high-level nature of our analysis and obfuscation of internal variable names, developers should always exercise caution when using a systemwide keystore.

It is also possible that some apps hardcode encryption keys after obfuscating the keys using Dexguard or a similar tool, but this was not possible to detect given our manual review is relatively cursory and intended primarily to verify our static analysis results and identify other APIs used.

8.2 Android Developer Survey Questions

1. Which of the following best describes your role?
    a. Programmer/Developer
    b. Software Tester/Quality Assurance
    c. Project Manager
    d. Software Design/Architecture
    e. Administration (Non-Technical)
    f. Other

2. Approximately how many people (including project managers, developers, testers, etc.) are involved in developing the app?
    a. 1
    b. 2 - 5
    c. 6 - 20
    d. 21 - 50
    e. 50+

3. How many years of experience do you have working with Android app development?
    a. None
    b. Less than 1 year
    c. 1 - 5 years
    d. 5 - 10 years
    e. 10 years or more

4. On a five-point scale, how much do you agree with the following statement: Our development team prioritizes security as part of the development process.
    a. Strongly agree
    b. Agree
    c. Neither agree nor disagree
    d. Disagree
    e. Strongly disagree

5. On a five-point scale, how much do you agree with the following statement: Our app collects and processes potentially sensitive user data (e.g., name, other demographic information, health data, financial data, etc.).
    a. Strongly agree
    b. Agree
    c. Neither agree nor disagree
    d. Disagree
    e. Strongly disagree
6. On a five-point scale, how much do you agree with the following statement: I am familiar with the concept of trusted hardware (e.g., Intel SGX and Arm TrustZone).
    a. Strongly agree
    b. Agree
    c. Neither agree nor disagree
    d. Disagree
    e. Strongly disagree
7. On a five-point scale, how much do you agree with the following statement: I am familiar with the Android Keystore trusted hardware API, commonly used in Android development for credential storage (e.g., storing cryptographic keys).
    a. Strongly agree
    b. Agree
    c. Neither agree nor disagree
    d. Disagree
    e. Strongly disagree
8. [*If app did not reference Android Keystore API at all.*] Based on our static analysis as of November 2023, your app was recorded as not using the Android Keystore API. Which of the following reasons best describe the main considerations behind this decision? Please select all that apply.
    - Security benefits were unclear
    - Security benefits were not needed given type of data (if any) collected by app
    - Performance concerns
    - Lack of features: Desired algorithm and/or key size was unavailable with Android Keystore

- We wanted to maximize our app's ability to run on many different devices (potentially running older versions of Android)
- App was developed prior to Android's Keystore API release date in 2013
- Found Keystore API difficult to use
- Unaware this API existed
- Don't know/don't remember
- We believe your static analysis result to be incorrect: [open text]
- Other: [open text]

9. [*If app did not reference Android Keystore API at all.*] To the best of your knowledge, what libraries, if any, does your app use within Android for credential storage (either user login credentials or developer credentials such as cryptographic keys)? [Open text]

10. [*If app was recorded as disabling StrongBox.*] A secure element (called the StrongBox Keymaster in Android) is a more advanced form of trusted hardware. Based on our static analysis of your app from November 2023, we determined your app used the Android Keystore API but disabled usage of the secure element in at least one instance. (Specifically, the setIsStrongBoxBacked API described in the link above was set to false). Which of the following reasons best describe the main considerations behind this decision? Please select all that apply.

    - Security benefits were unclear
    - Security benefits were not needed given type of data (if any) collected by app
    - Performance concerns
    - Lack of features: desired cryptographic algorithm and/or key size was unavailable with StrongBox Keymaster
    - We wanted to maximize our app's ability to run on many different devices (potentially running older versions of Android)
    - Don't know/don't remember
    - We believe your static analysis result to be incorrect: [open text]
    - Other: [open text]